НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

«На правах рукопису»
УДК 004.043

«До захисту допущено»

Завідувач кафедри
_____ І.Р. Пархомей
(підпис)
“___”_____2018 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Розробка планувальника завдань для розподілених Enterprise систем»

Виконав: студент другого курсу, групи ІТ-74мп
(шифр групи)

Хижняк Микита Дмитрович _____ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник доцент, к.т.н., Пасько В.П. _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

Рівень вищої освіти – другий (магістерський)

Спеціальність 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ І.Р. Пархомей
(підпис)
«___»_____2018 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Хижняку Микиті Дмитровичу
(прізвище, ім'я, по батькові)

1. Тема дисертації «Розробка планувальника завдань для розподілених Enterprise систем»,
науковий керівник дисертації _____доц., к.т.н. Пасько В.П._____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 2018 р. №_____

2. Термін подання студентом дисертації _____

3. Об'єкт дослідження системи планування завдань в РСОД, засоби та методи планування у розподілених системах.

4. Предмет дослідження методи та засоби планування завдань, які можуть використовуватись або використовуються в системах планування для РСОД

5. Перелік завдань, які необхідно розробити — аналіз існуючих методів планування завдань обробки даних в РСОД; розробка математичної моделі планування завдань в РСОД; розробка методу планування завдань в РСОД; рішення проблеми пошуку найближчих завдань з урахуванням різнорідності атрибутів і їх значимості для ресурсоспоживання; розробка методики планування завдань в РСОД; розробка методу синхронізації

результатів виконання завдань на різних вузлах розподіленої системи. Забезпечення коректності роботи планувальника у розподіленій системі; програмна реалізація методів планування завдань у РСОД у вигляді планувальника завдань для розподілених Enterprise систем.

6. Орієнтовний перелік ілюстративного матеріалу 1.Схема структурна варіантів використання. 2.Схема структурна діяльності. 3.Схема структурна послідовності. 4. Схема моделі бази даних. 5. Схема структурна класів інтеграції з БД. 6.Схема структурна класів програмного забезпечення. 7.Схема структурна компонентів. 8.Схема структурна розгортування.

7. Орієнтовний перелік публікацій

1. Хижняк М.Д. Алгоритми балансування мережевого навантаження та проксінг. Порівняльна характеристика типів топологій балансування/ Микита Дмитрович Хижняк. // Актуальные научные исследования в современном мире. – 2018. – №9(41).

2.Хижняк М.Д, Ромащенко П.С. Міжнародна конференція Modern scientific challenges and trends Warsaw – The use of self-organizational neural network for planning tasks in distributed systems.

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9.Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Вивчення рекомендованої літератури		
2.	Аналіз існуючих методів розв’язання задачі		
3.	Постановка та формалізація задачі		
4.	Розробка інформаційного забезпечення		
5.	Алгоритмізація задачі		
6.	Обґрунтування використовуваних технічних засобів		
7.	Розробка програмного забезпечення		
8.	Налагодження програми		
9.	Виконання графічних документів		
10.	Оформлення пояснювальної записки		

11.	Подання дисертації на попередній захист		
12.	Подання дисертації на основний захист		
13.	Подання дисертації рецензенту		

Студент _____ Хижняк М.Д
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ Пасько В.П.
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

У роботі розглянуто проблему роботи планувальників завдань у розподілених Enterprise системах. Об’єктом дослідження є системи планування завдань в РСОД, засоби та методи планування у розподілених системах. В роботі показано основні особливості існуючих алгоритмів планування та їх програмних реалізацій, їх переваги та недоліки. Розроблено планувальник завдань для розподілених Enterprise систем. Цілі розробки – підвищення швидкості виконання завдань у розподілених системах, зменшення часу очікування завдань у черзі на виконання за рахунок їх обробки різними вузлами системи. Задачі розробки – це створення програми-планувальника завдань (розрахунків, обробки даних, тощо) у розподілених Enterprise системах, що буде дозволяти виконувати завдання на декількох серверах застосувань за наявності декількох серверів баз даних та проводити синхронізацію результатів своєї роботи.

В рамках дослідження використовується теорія алгоритмів та математична статистика в якості методів дослідження. Математична модель та запропонований метод планування завдань у РСОД є науковою новизною дослідження, що отримали відображення у архітектурі планувальника завдань для Enterprise систем з розподіленою архітектурою. Планувальник є open-source бібліотекою, що може бути впроваджена у будь-яку систему для вирішення задачі планування та виконання завдань у фоновому режимі.

Ключові слова: планування, балансування, планувальник, обробка даних, Enterprise системи, синхронізація обчислень, математична модель, теорія алгоритмів.

Розмір пояснювальної записки – 98 аркушів, містить 25 рисунків, 21 таблицю, 21 джерело, 8 додатків.

ABSTRACT

Examines the problem of work scheduler tasks in distributed Enterprise systems. The object of research is the system of planning tasks in the RSD, the means and methods of planning in distributed systems. The main features of existing planning algorithms and their program implementations, their advantages and disadvantages are shown in the work.

Developed a Task Scheduler for Distributed Enterprise Systems. Development goals - increasing the speed of tasks in distributed systems, reducing the waiting time in the queue for execution due to their processing of different nodes of the system. The development tasks are the creation of a Task Scheduler (calculations, data processing, etc.) in distributed Enterprise systems that will allow tasks to run on multiple application servers in the presence of multiple database servers and synchronize their work results.

The research uses the theory of algorithms and mathematical statistics as research methods. The mathematical model and the proposed method of planning tasks in the RSD is a scientific novelty of research, reflected in the architecture of the task scheduler for Enterprise systems with distributed architecture. The Scheduler is an open-source library that can be implemented in any system to solve the task of scheduling and performing tasks in the background.

Keywords: scheduling, balancing, scheduler, data processing, enterprise system, synchronization of computations, mathematical model, algorithm theory.

Explanatory note size - 98 sheets, contains 25 illustrations, 21 tables, 21 sources, 8 applications.

Пояснювальна записка
до магістерської дисертації

на тему: «Розробка планувальника завдань для розподілених Enterprise систем»

Київ – 2018 року

ЗМІСТ

ПЕРЕЛІК ТЕРМІНІВ ТА СКОРОЧЕНЬ 4
ВСТУП 6
1 АКТУАЛЬНІСТЬ ТА ПОСТАНОВКА ЗАДАЧІ 8
1.1 Предметна область 8
1.2. Способи управління задачами 9
1.2.2. Управління пам'яттю 10
1.2.3. Управління доступом, синхронізація 10
1.3 Види паралельних обчислювальних систем 11
1.3.1 Мультипроцесорна система 11
1.3.2 Мультикомпьютерна система 12
1.3.3 Розподілені системи 12
1.4 Постановка задачі 13
2 ІСНУЮЧІ РЕАЛІЗАЦІЇ ПЛАНУВАЛЬНИКІВ ЗАДАЧ 16
2.1 Характеристики планувальників задач обчислюваних систем 16
2.2 Реалізації планувальників задач 18
2.2.1 NCron 18
2.2.2 Hangfire 19
2.2.3 Quartz 20
2.2.4. FluentScheduler 22
3 РОЗРОБКА АЛГОРИТМУ РОЗПОДІЛУ ЗАДАЧ В ПЛАНУВАЛЬНИКУ 25
3.1 Методи планування та управління задачами 25
3.1.1 Rate-monotonic 25
3.1.2 Deadline Monotonic 26
3.1.3 Метод фонового виконання 26
3.1.4 Метод опитування 27
3.1.5 Алгоритм невідкладного сервера 27
3.1.6 Алгоритм last chance 27
3.1.7 EDF 28
3.1.8 Сервер, що допускає затримку і алгоритм обміну пріоритетами 28
3.2 Математична модель задачі 29
3.3 Алгоритм NFDH 30
3.4 Подання набору масштабованих завдань 31
3.5 Опис генетичного алгоритму 31
4 РЕАЛІЗАЦІЯ ГЕНЕТИЧНОГО АЛГОРИТМУ 37
4.1 Визначення вимог і завдань 37

4.2 Опис функціоналу бібліотеки 37
4.3 Прецеденти 39
4.3.1 Постановка задач на виконання за допомогою планувальника 41
4.3.2Налаштування роботи планувальника у архітектурі web-ферми 42
4.3.3 Налаштування кожного екземпляру планувальника 44
4.4 Архітектура планувальника 44
4.4.1 Монолітна та мікро-сервісна архітектура 44
4.4.2 Загальна архітектура планувальника 47
4.4.3 Схема мікро-сервісів планувальника 47
4.5 Архітектурні шаблони роботи з базою даних у мікро-сервісах 50
4.5.1 Shared database 50
4.5.2 Database per service 51
4.6 Вибір технологій та їх обґрунтування 53
4.6.1 Вибір основної платформи для бібліотеки 53
4.6.2 Вибір архітектури розгортування 56
4.6.3 Вибір мови програмування 57
4.6.4 Вибір допоміжних бібліотек 57
4.7 Реалізація бібліотеки та її компонентів 59
4.7.1 Scheduling та Balancing мікро-сервіси 60
4.7.2 Job мікро-сервісу 64
4.7.3 API взаємодії з планувальником для систем-клієнтів 65
4.7.4 Структура системних таблиць бази даних планувальника 67
5 РОЗРОБЛЕННЯ СТАРТАП ПРОЕКТУ 70
5.1 Опис ідеї проекту 70
5.2 Технологічний аудит ідеї проекту 72
5.3 Аналіз ринкових можливостей запуску стартап-проекту 73
5.4 Розроблення ринкової стратегії проекту 77
5.5 Розроблення маркетингової програми стартап-проекту 78
6 МОДЕЛЮВАННЯ АЛГОРИТМУ ПЛАНУВАННЯ 81
6.1 Моделювання та генерація набору завдань 81
6.2 Результати моделювання 81
6.3 Випробування бібліотеки планувальника 84
ВИСНОВКИ 89
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 91

ПЕРЕЛІК ТЕРМІНІВ ТА СКОРОЧЕНЬ

Фреймворк	Інфраструктура програмних рішень, що полегшує розробку складних систем.
Use-Case	(з англ. варіант використання) відображає варіанти взаємодії користувача із програмою.
API	Інтерфейс прикладного програмування, опис методів (набір класів, процедур, функцій, структур або констант), за допомогою яких одна комп'ютерна програма може взаємодіяти з іншою програмою.
Бібліотека	Перелік просторів імен, класів та їх властивостей і функцій, що об’єднані в одну dll бібліотеку для поставки клієнтам.
Open-source	Відкрите програмне забезпечення. Альтернативна назва вільного програмного забезпечення, введена через неоднозначність вираження "вільного програмного забезпечення" англійською мовою. Вираження означає доступність творів та матеріалів, використаних для його створення, за вільною або відкритою ліцензією.
Cross-platform	У комп'ютерних програмах крос-платформене програмне забезпечення (також багатоплатформене програмне забезпечення або незалежне від платформи програмне забезпечення) це комп'ютерне програмне забезпечення, яке реалізується на кількох обчислювальних платформах.
Middleware	Супровідне програмне забезпечення (англ. Middleware; також перекладається як проміжне програмне забезпечення, програмне забезпечення середнього шару) це широко використовуваний термін, означаючий шар або комплекс технологічного забезпечення для

	забезпечення взаємодії між різними додатками, системами, компонентами.
Non-preemptive multitasking	Кооперативна багатозадачність, також відома як багатозадачність без витіснення, це стиль комп'ютерної багатозадачності, в якому операційна система ніколи не ініціює перемикання контексту від запущеного процесу до іншого процесу. Замість того, процеси періодично добровільно поступаються контролем , або перебувають в режимі очікування, для того щоб кілька додатків мали можливість працювати одночасно.
Preemptive multitasking	Витісняюча багатозадачність це вид багатозадачності, при якій операційна система приймає рішення про переключення між задачами після закінчення якогось кванту часу.

ВСТУП

У наш час комп’ютерні системи займають чи не найвагоміше місце у процесі життєдіяльності суспільства. З кожним роком їх потужність та кількість користувачів зростає і з’являються все нові способи їх використання для різноманітних задач. Користувачі у свою чергу стають вибагливішими і прагнуть до автоматизації більшої частини процесів задля економії власного часу, адже це і є найцінніший ресурс. Одним з таких процесів є обробка даних та обчислення.

Зараз у світі комп’ютерні системи Enterprise рівня, тобто такі що застосовуються для керування бізнес-процесами великих компаній, є дуже поширеними. Однією з головних задач таких систем є розподілені обчислення та обробка даних, що відбувається за певним розкладом.

Системи планування завдань є дуже відомим та ефективним рішенням проблеми гнучкого та швидкого призначення завдань обробки даних на доступних обчислювальних ресурсах розподілених систем обробки даних (РСОД). Як правило РСОД мають декілька властивостей, які підвищують складність розробки та використання розподілених систем планування завдань, насамперед це такі властивості як:

1. Різнорідність ресурсів обчислювальних вузлів, що призводить до ускладнення алгоритмів розподілення та планування завдань;
2. Відсутність початкових даних про ресурсоємність завдань, що також ускладнює алгоритми планування та розподілення [1].

Актуальність теми дослідження. Розповсюдження грід-систем, хмарних технологій, кластерних систем призводить до збільшення кількості задач планування у розподілених системах. Час, відведений на вирішення обчислювального завдання постійно зменшується. Всі ці фактори обґрунтовують актуальність розробки нових методів та засобів планування завдань в РСОД. В останні декілька років розвиток обчислювальних хмарних ресурсів приводить до створення все більшої кількості великих Enterprise систем, що розміщуються повністю на обчислювальних потужностях у хмарі, а саме тому є розподіленими.

Таким чином задача використання планувальників завдань у розподілених Enterprise системах потребує точного алгоритму не лише розподілу завдань між

вузлами системи але і синхронізацію результатів обчислень або обробки даних з різних вузлів системи. Об'єктом дослідження є системи планування завдань в РСОД, засоби та методи планування у розподілених системах. Предметом дослідження виступають методи та засоби планування завдань, які можуть використовуватись або використовуються в системах планування для РСОД.

Постановка проблеми. Для досягнення мети необхідно вирішити наступні завдання:

1. Аналіз існуючих методів планування завдань обробки даних в РСОД.
2. Розробка математичної моделі планування завдань в РСОД.
3. Розробка методу планування завдань в РСОД.
4. Рішення проблеми пошуку найближчих завдань з урахуванням різнорідності атрибутів і їх значимості для ресурсоспоживання.
5. Розробка методики планування завдань в РСОД.
6. Розробка методу синхронізації результатів виконання завдань на різних вузлах розподіленої системи. Забезпечення коректності роботи планувальника у розподіленій системі.
7. Програмна реалізація методів планування завдань у РСОД у вигляді планувальника завдань для розподілених Enterprise систем.

В рамках дослідження використовується теорія алгоритмів та математична статистика в якості методів дослідження. Математична модель та запропонований метод планування завдань у РСОД є науковою новизною дослідження, що отримали відображення у архітектурі планувальника завдань для Enterprise систем з розподіленою архітектурою. Планувальник є open-source бібліотекою, що може бути впроваджена у будь яку систему для вирішення задачі планування та виконання завдань у фоновому режимі.

Постановка задачі. Метою роботи є розробка методів та засобів скорочення часових витрат на виконання завдань в РСОД, а також реалізація засобів покращення обробки та планування завдань у системах з розподіленою архітектурою. Забезпечення коректності роботи планувальника завдань у розподіленій системі, а саме коректність розподілу завдань між вузлами системи, а також коректність синхронізації результатів виконання завдань на різних вузлах розподіленої системи.

1 АКТУАЛЬНІСТЬ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Предметна область

Так як кількість і складність сучасних веб-додатків продовжують зростати, зростає і складність всіх компонентів додатків, що знаходяться в їх основі. Планування задач є загальною вимогою для додатків у сучасних системах, і тому це є предметом постійної турботи для розробників. Хоча сучасні технології планування задач були вдосконалені в порівнянні з примітивними тригерами баз даних і окремими потоками планувальників, планування задач залишається нетривіальною проблемою.

Одним з найбільш ефективних рішень даної проблеми є фреймворки та бібліотеки планування з власним API, що розробляються багатьма компаніями та як частина open-source проектів. Планувальник представляє собою середовище розробки для планування задач із відкритим (або закритим) вихідним кодом, який забезпечує простий, але ефективний механізм планування задач у додатках на певній платформі, або у cross-platform додатках.

Планувальник задач планує і координує задачі під час виконання. Задача це одиниця роботи, що виконує конкретні дії. Зазвичай різні задачі можуть виконуватися паралельно. Як приклад задачі можна привести роботу, виконувану елементами групи задач, паралельними алгоритмами і асинхронними агентами.

Планувальник задач здійснює управління подробицями задач для ефективного планування на комп'ютерах з великим обсягом обчислювальних ресурсів. Планувальник задач також використовує новітні можливості базової операційної системи. Таким чином, функції, які залежать від середовища виконання з паралелізмом, автоматично масштабуються і вдосконалюються обладнанням з розширеними можливостями.

Планувальники задач представляють розробникам можливість планувати задачі за часовим інтервалом або за добу. В ньому реалізуються

відносини типу "багато-до-багатьох" для задач і тригерів, що дозволяє об'єднати множинні задачі з різними тригерами [1]. Додатки у які, входить планувальник у якості модуля, можуть повторно використовувати задачі з різних подій і групувати численні задачі для однієї події.

Зазвичай планувальники налаштовуються за допомогою конфігураційних файлів в якому вказуються такі основні компоненти будь якого планувальника як:

1. джерело даних для транзакцій з середовищем зберігання джобів планувальника;
2. посилання на глобальних підписників планувальника;
3. посилання на тригери, плагіни, потоки пулів.

При цьому сучасний планувальник не має інтегруватися з контекстом або посиланнями application сервера. Таким чином планувальник не має отримувати доступ до внутрішніх функцій Web-сервера і не заважають джерелам сервера даних і кешування, не впливаючи на роботу інших модулів (мікро-сервісів застосування).

1.2 Способи управління задачами

1.2.1 Керування часом

Одним з основних властивостей систем з планувальниками є їхня здатність ізолювати один від одного задачі, тому якщо в програмі виникає збій або виконуються якісь виключні (помилкові) операції, система може швидко блокувати програму, ініціювати відновлення і захист інших задач або самої системи від серій шкідливих команд .

Якщо не виконується обробка критичних ситуацій або вона відбувається недостатньо швидко, система перериває операцію і блокує її, щоб не постраждала надійність і готовність решти системи.

Особливу важливість набувають такі інструменти як засоби роботи з таймерами, необхідні для систем з жорстким тимчасовим регламентом.

Розвиненість цих можливостей це необхідний атрибут систем з планувальниками. Вони, як правило, дозволяють:

вимірювати і ставити різні часові відтинки;

генерувати переривання після закінчення тимчасових інтервалів;

створювати разові і циклічні таймери.

1.2.2 Управління пам'яттю

Система з планувальником повинна вміти управляти пам'яттю в залежності від критичності задач. Для стійкої роботи процесів потрібні механізми виділення пам'яті при їх породженні, використання пам'яті при життєдіяльності і звільнення.

Системи з планувальниками дозволяють програмістам ізолювати спільно використовувані бібліотеки, дані і системне програмне забезпечення, а також задачі. Той самий захист запобігає переповненню стеків пам'яті, що викликається діями будь-яких програм [2].

1.2.3 Управління доступом, синхронізація

При одночасній роботі декількох задач в системі з планувальником, система повинна забезпечити стійкий механізм для обміну інформацією між запущеними задачами. Зв'язок між задачами (Intasks communication) є ключем до розробки систем з планувальниками, та самих планувальників як сукупності процесів, в яких кожен процес виконує відведену йому частину спільного завдання.

Для операційних систем з планувальниками характерна розвиненість intertasks механізмів. До таких механізмів відносяться: семафори, події, сигнали, засоби для роботи з пам'яттю, канали даних (pipes), черги повідомлень. Багато з подібних механізмів використовують і в операційних системах загального призначення, але їх реалізація в системах з планувальниками має свої особливості, оскільки час виконання задач майже не залежить від стану системи,

і в кожній системі з планувальником є принаймні один швидкий механізм передачі даних від задачі до задачі.

1.3 Види паралельних обчислювальних систем

Для виконання складних розрахунків і ефективної роботи великих обчислювальних систем зазвичай використовують одну з наступних моделей:

мультипроцесорна система;

мультикомпьютерна система;

розподілена система.

1.3.1 Мультипроцесорна система

Для мультипроцесорної системи характерна наявність спільної пам'яті, доступ до якої може отримати будь-який процесор, що входить до її складу (рис. 1.1). У подібній системі спільну пам'ять використовують для обміну інформацією між різними процесорами. Це збільшує швидкість обміну даними, але сама реалізація спільної пам'яті, якою будуть одночасно користуватися сотні і сотні різних процесорів, на практиці куди більш складний і трудомісткий процес, ніж здійснення міжпроцесорної взаємодії через додаткові модулі [3].

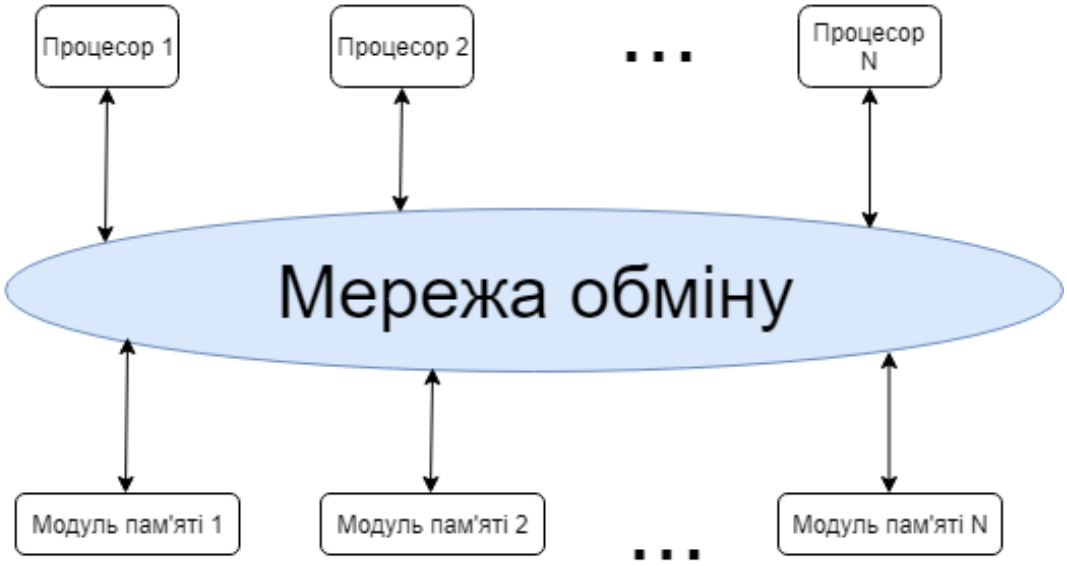


Рисунок 1.1 Загальна архітектура мультипроцесорної системи

1.3.2 Мультикомпьютерна система

На відміну від цього, мультикомпьютерна система не має спільної пам'яті, але ж кожен з процесорів має невелику ділянку локальної пам'яті, де і відбувається зберігання оброблюваної і аналізованої їм інформації. Подібна архітектура означає, що жоден з процесорів не зможе отримати безпосереднього доступу до даних іншого, і це дещо ускладнює обмін інформацією вже на етапі поєднання результатів роботи. Для здійснення подібного трафіку даних між процесами існує додатковий елемент, так звана підсистема «внутрішньої комунікаційної мережі» [3].

Незважаючи на те, що мультикомпьютерна система дещо поступається в швидкості обробки мультипроцесорній, реалізація такої системи значно простіше і дешевше (при аналогічному числі використовуваних процесорів).

1.3.3 Розподілені системи

Розподіленою обчислювальною системою (РОС) можна вважати набір з'єднаних каналами зв'язку незалежних комп'ютерів, які для стороннього користувача є єдиним цілим. РОС можна уявити, як кілька окремих персональних комп'ютерів або серверів, об'єднаних в єдину систему за допомогою мережі Інтернет. Для забезпечення роботи обладнання РОС у вигляді єдиного цілого, стек програмного забезпечення (ПЗ) зазвичай розбивають на два шари. На верхньому шарі розташовуються розподілені додатки, що відповідають за вирішення певних прикладних задач засобами РОС. їх функціональні можливості базуються на нижньому шарі - проміжному програмному забезпеченні (ППЗ). ППЗ взаємодіє з системним ПЗ і мережевим рівнем, для забезпечення прозорості роботи додатків в РОС (рис. 1.2).

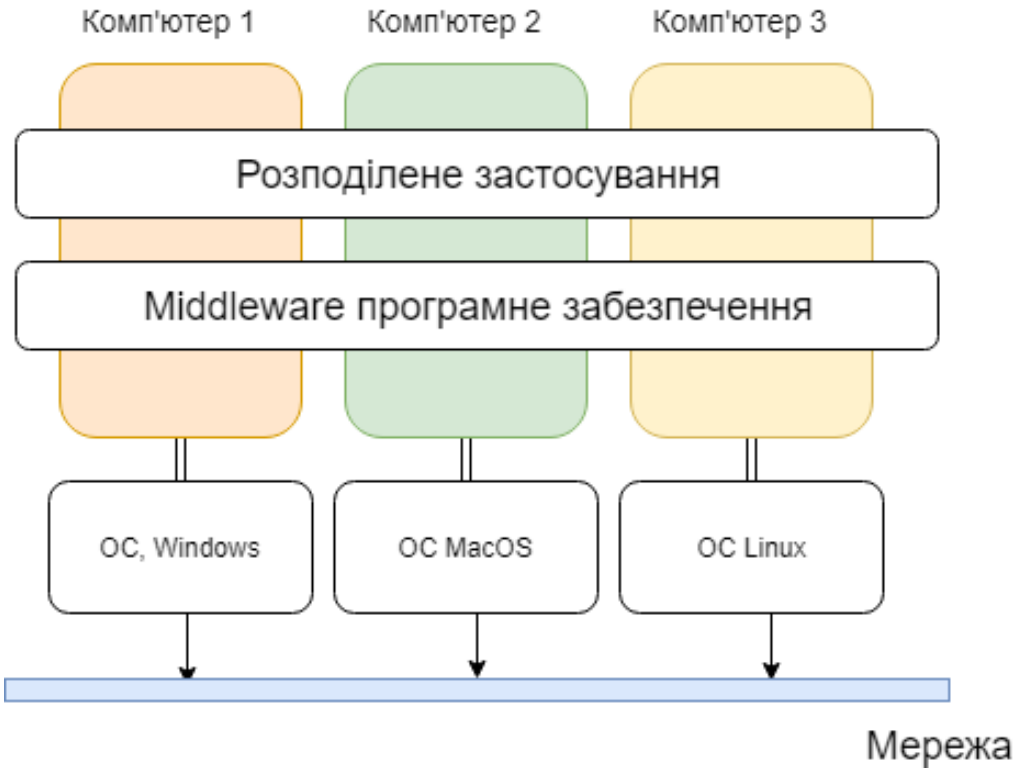


Рисунок 1.2 Ієрархія програмного забезпечення у розподілених системах

Природно, обмін даними через Інтернет, як і самі способи організації подібного повідомлення (обмін пакетами, маршрутизація і ін.) між машинами, що знаходяться в різних частинах земної кулі (фізично), не можна назвати стабільним і виразно швидким, а тому часові показники подібних систем досить низькі. Однак РОС відрізняються високою надійністю, з огляду на те, що вихід з ладу одного елемента або вузла мережі не грає особливої ролі в її працездатності. До того ж, вони мають практично необмежений потенціал в плані нарощування продуктивності, чим більше комп'ютерів буде об'єднано цією структурою, тим вище буде їх загальна обчислювальна потужність.

1.4 Постановка задачі

Актуальність теми дослідження. Розповсюдження грід-систем, хмарних технологій, кластерних систем призводить до збільшення кількості задач планування у розподілених системах. Час, відведений на вирішення обчислювальної задачі постійно зменшується. Всі ці фактори обґрунтовують актуальність розробки нових методів та засобів планування задач в РСОД. В останні декілька років розвиток обчислювальних хмарних ресурсів приводить до

створення все більшої кількості великих Enterprise систем, що розміщуються повністю на обчислювальних потужностях у хмарі, а саме тому є розподіленими. Таким чином задача використання планувальників завдань у розподілених Enterprise системах потребує точного алгоритму не лише розподілу задач між вузлами системи але і синхронізацію результатів обчислень або обробки даних з різних вузлів системи.

Постановка проблеми. Для досягнення мети необхідно вирішити наступні завдання:

1. Аналіз існуючих методів планування завдань обробки даних в РСОД.
2. Розробка математичної моделі планування завдань в РСОД.
3. Розробка методу планування завдань в РСОД.
4. Рішення проблеми пошуку найближчих завдань з урахуванням різнорідності атрибутів і їх значимості для ресурсоспоживання.
5. Розробка методики планування завдань в РСОД.
6. Розробка методу синхронізації результатів виконання завдань на різних вузлах розподіленої системи. Забезпечення коректності роботи планувальника у розподіленій системі.
7. Програмна реалізація методів планування завдань у РСОД у вигляді планувальника завдань для розподілених Enterprise систем.

Постановка задачі. Метою роботи є розробка методів та засобів скорочення часових витрат на виконання задач в РСОД, а також реалізація засобів покращення обробки та планування задач у системах з розподіленою архітектурою. Забезпечення коректності роботи планувальника задач у розподіленій системі, а саме коректність розподілу задач між вузлами системи, а також коректність синхронізації результатів виконання задач на різних вузлах розподіленої системи. Поставлену задачу доцільно розділити на структурні частини:

1. Аналіз існуючих рішень планування завдань у розподілених системах.
2. Розгляд загальних алгоритмів розподілу та планування задач.
3. Розробка алгоритму ефективного планування та синхронізації результатів виконання завдань у розподілених Enterprise системах.

ВИСНОВКИ ДО РОЗДІЛУ

В даному розділі біло проведено дослідження предметної області сфери планувальників задач у розподілених системах та її актуальності у наш час. У рамках дослідження розглянуто основні особливості управління задачами:

керування часом;

управління пам’яттю;

управління доступом (синхронізація).

Визначено, що способи виконання задач у системах мають свою специфіку та включають в себе створення алгоритмів поділу ресурсів системи, планування їх незалежного виділення і звільнення для задач системи.

Детально розглянуто предметну область розподілених систем, як об’єкта, напряму пов’язаного із темою дослідження. Проаналізовано основні типи розподілених систем:

Мультипроцесорна система, як система з наявністю спільної пам'яті, доступ до якої може отримати будь-який процесор, що входить до її складу;

Мультикомп’ютерна система, як система, що не має спільної пам'яті, але ж кожен з процесорів має невелику ділянку локальної пам'яті, де і відбувається зберігання оброблюваної і аналізованої їм інформації;

Розподілена система, як система, що являє собою набір з'єднаних каналами зв'язку незалежних комп'ютерів, які для стороннього користувача є єдиним цілим;

Для роботи поставлено задачу створення планувальника задач у розподілених системах, для скорочення часових витрат на виконання задач в РСОД, а також реалізація засобів покращення обробки та планування задач у системах з розподіленою архітектурою.

2 ІСНУЮЧІ РЕАЛІЗАЦІЇ ПЛАНУВАЛЬНИКІВ ЗАДАЧ

2.1 Характеристики планувальників задач обчислюваних систем

Спосіб вирішення питання про планування задач істотно залежить від того, незалежні або залежні (пов'язані один з одним) задачі в списку, а тому їх можна в цілому розбити на дві великі категорії:

планування незалежних задач;

планування залежних задач.

На даний момент існує безліч різних алгоритмів планування задач, які переслідують різні цілі і забезпечують різну якість мультизадачної обробки всередині системи. Серед цієї безлічі алгоритмів розглянемо докладніше дві групи алгоритмів, що зустрічаються найбільш часто: алгоритми, засновані на квантуванні, і алгоритми, засновані на пріоритетах. Базова класифікація алгоритмів планування зображена на рис. 2.1.

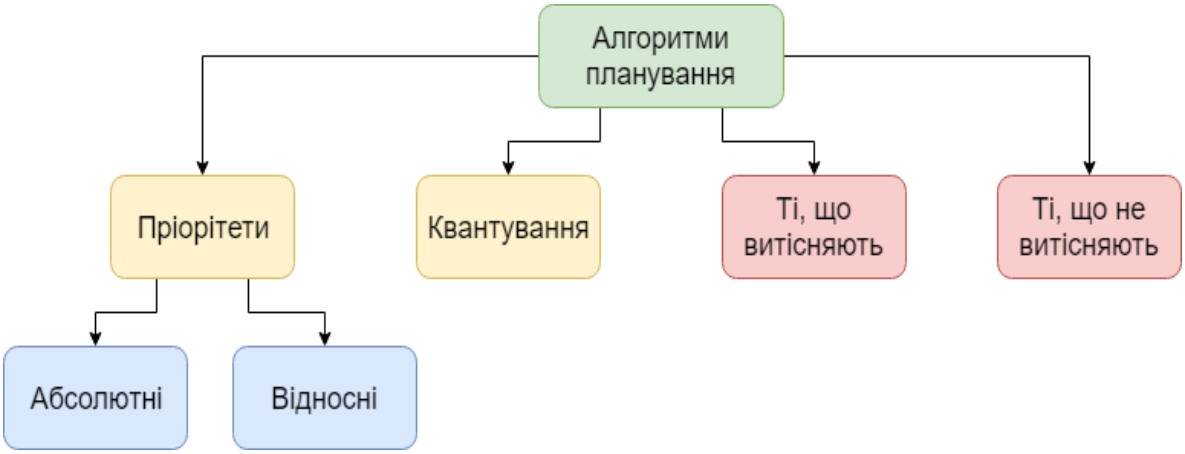


Рисунок 2.1 Базова класифікація алгоритмів планування

Алгоритми, засновані на ідеї квантування, здійснюють зміну активної задачі, якщо відбувається одне з наступного:

1. задача завершилася і залишила систему;
2. виникла помилка;
3. задача перейшла в стан ОЧІКУВАННЯ;
4. вичерпаний квант процесорного часу, відведений даній задачі.

Задача, що вичерпала свій квант, переводиться в стан ГОТОВНІСТЬ і очікує, коли їй буде надано новий квант процесорного часу, а на виконання

відповідно до визначеного правила вибирається нова задача з черги готових. Таким чином, жодна задача не займає процесор(систему виконання) надовго, тому квантування широко використовується в системах поділу часу [4].

Самі кванти можуть відрізнятися для різних задач або ж взагалі змінюватися з плином часу. Задачі, які в повному обсязі використовували виділений їм для роботи час (наприклад, через відхід на виконання операцій введення-виведення), часто отримують в системі додаткові привілеї при подальшому обслуговуванні. По-різному може бути організована і сама черга готових задач: циклічно, за правилом "перший прийшов - перший обслугувався" (FIFO), "останній прийшов - перший обслугувався" (LIFO) або за допомогою іншого, більш складного методу.

Іншим способом класифікації алгоритмів планування можна вважати поділ їх на ті, що витісняють і ті, що невитісняють.

Non-preemptive multitasking (невитісняюча багатозадачність), це спосіб планування задач, при якому активна задача виконується до тих пір, поки вона сама, за власною ініціативою, не віддасть управління планувальником для того, щоб той вибрав з черги іншу, готоу до виконання задачу.

Preemptive multitasking (витісняюча багатозадачність), це такий спосіб, при якому рішення про переключення задачі з виконання однієї задачі на виконання іншої задачі приймається планувальником, а не найактивнішою задачею.

Основною відмінністю між preemptive і non-preemptive варіантами багатозадачності є ступінь централізації механізму планування задач. При витісняючій багатозадачності даний механізм цілком зосереджений в обчислювальній системі, і програміст пише свій додаток, не піклуючись про те, як задача буде виконуватися паралельно з іншими задачами. При цьому сама система виконує наступні функції: визначає момент зняття з виконання активної задачі, запам'ятовує її контекст, вибирає з черги готових задач наступну і запускає її на виконання, завантажуючи її контекст.

При невитісняючій багатозадачності механізм планування розподілений між системою і прикладними програмами. Прикладна програма, отримавши

управління, сама визначає момент завершення своєї чергової ітерації і повертає управління системі за допомогою будь-якого системного виклику, а та, в свою чергу, формує черги задач і вибирає відповідно до деякого алгоритму (наприклад, з урахуванням пріоритетів) наступне завдання на виконання. Такий механізм створює проблеми як для користувачів, так і для розробників [5].

2.2 Реалізації планувальників задач

Розглянемо готові реалізації планувальників задач для різних платформ та систем-клієнтів.

2.2.1 NCron

NCron - потужний планувальник / скриптер / менеджер автоматизації. Має маленький розмір але широкий спектр можливостей. Перелік задач NCron:

запускати довільні програми як сервіси;

запускати задачі "від імені" вказаних користувачів;

відслідковувати і перезапускати прострочені задачі і нагадування;

здатний працювати з функціями API ядра операційної системи;

nnCron здатний відслідковувати файли, прапори, час простою.

NCron дозволяє використовувати в задачах скриптові мови VBScript і JScript, а також регулярні вирази. Може виконувати довільні програми на мові Forth, розширюється за рахунок плагінів і т.д.

NCron розуміє cron-формат (Unix) і управляється за допомогою текстових кронтаб-файлів. Зберігає налаштування і данні в текстових файлах.

NCron має графічну оболонку з якої можна видаляти / додавати / редагувати і запускати задачі, встановлювати нагадування, змінювати налаштування програми. Крім цього NCron має розвинені можливості по роботі з ключами командного рядка.

NCron може бути запущений в якості служби (сервісу) або як звичайна програма.

Приклад використання Jscript у NCron:

```
#( jscript-calc
Time: 0 12 * * * *
Action:
<JScript>
var WshShell = WScript.CreateObject("WScript.Shell");
WshShell.Run("calc");
WScript.Sleep(100);
WshShell.AppActivate("Calculator");
WScript.Sleep(100);
WshShell.SendKeys("2{+}2{*}3=");
</SCRIPT>
)#
```

Приклад використання NCron API C#:

```
public class MyJob : NCron.CronJob {
    public override void Execute() {
          System.IO.File.Copy(@"c:\\output.out",
          @"f:\\output.out");
     }
}
```

Переваги:

зручне та гнучке API для розгортування задач;

можливість встроювати розклади задач у збірку проекту, або ж зберігати їх у базі даних чи конфігураційних файлах;

гнучке та зручне API журналювання задач;

підтримка IoC контейнерів при розробці.

Недоліки:

підтримка лише Windows платформи;

неможливість розгорнути планувальник у якості контейнера;

не підтримує роботу у розподілених системах;

відсутність широкого набору засобів контролю за алгоритмом планування.

2.2.2 Hangfire

Hangfire – багатопотоковий і масштабований планувальник задач, побудований на клієнт-серверній архітектурі (рис. 2.2) на стеку технологій .NET

(в першу чергу Task Parallel Library і Reflection), з проміжним зберіганням задач в БД. Повністю функціональний в безкоштовній (LGPL v3) версії з відкритим вихідним кодом [5].

Переваги:

зручне та гнучке API для розгортування задач;

підтримка декількох ОС;

підтримка контейнеризації при розгортуванні та розробці.

Недоліки:

не підтримує роботу у розподілених системах;

складна архітектура планувальника, що потребує додаткових вкладень;

відсутність широкого набору засобів контролю за алгоритмом планування.

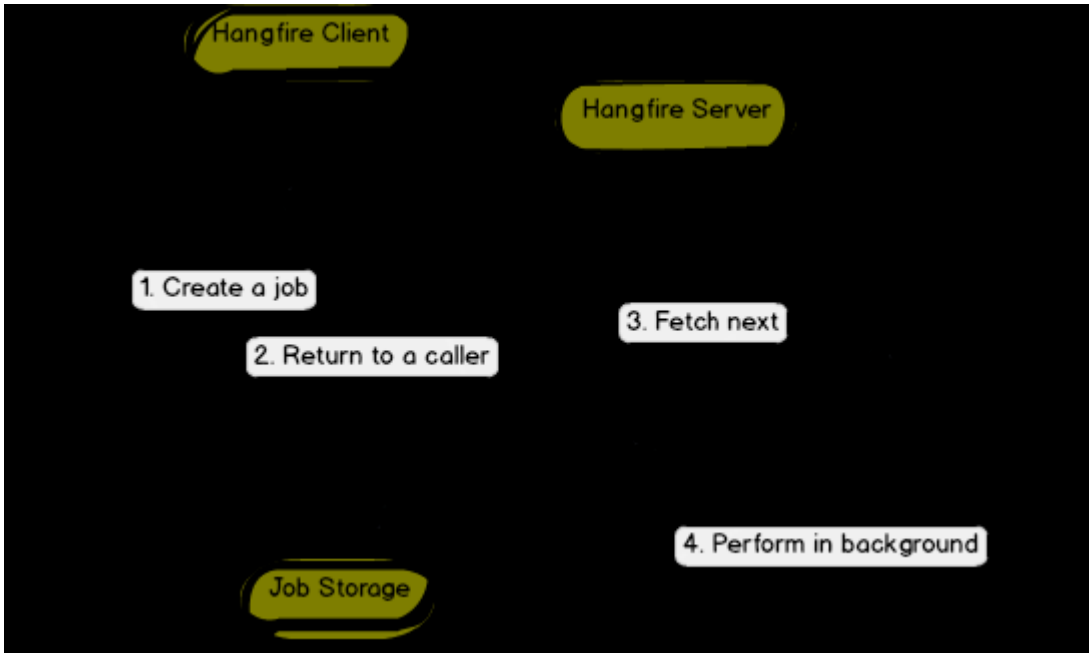


Рисунок 2.2 Клієнт серверна архітектура Hangfire з hangfire.io

2.2.3 Quartz

Quartz.NET - порт планувальника Quartz зі світу Java. Quartz.NET вирішує схожі завдання, як і Hangfire, підтримує довільну кількість «клієнтів» (додавання задачі) і «серверів» (виконання задачі), що використовують загальну БД. Але процес виконання різний.

Поділу на клієнтську і серверну частину в проекті немає - Quartz.NET поширюється у вигляді єдиної DLL. Для того, щоб конкретний екземпляр додатка дозволяв тільки додавати задачі, а не виконувати їх, необхідно його налаштувати.

Quartz.NET повністю безкоштовний, «з коробки» пропонує зберігання задач як in-memory, так і з використанням багатьох популярних СУБД (SQL Server, Oracle, MySQL, SQLite і т.п.). Зберігання in-memory є по-суті звичайним словником в пам'яті одного єдиного процесу-сервера, що виконує задач. Реалізувати кілька процесів-серверів стає можливим тільки при збереженні задач в БД. Для синхронізації, Quartz.NET не покладається на специфічні особливості реалізації конкретної СУБД (ті ж Application Lock в SQL Server), а використовує один узагальнений алгоритм.

Приклад роботи з C# API Quartz .NET:

```
IScheduler scheduler = await
StdSchedulerFactory.GetDefaultScheduler();
await scheduler.Start();

IJobDetail job = JobBuilder.Create<EmailSender>().Build();

ITrigger trigger = TriggerBuilder.Create()
  .WithIdentity("trigger1", "group1")
  .StartNow()
  .WithSimpleSchedule(x => x
        .WithIntervalInMinutes(1)
        .RepeatForever())
  .Build();

await scheduler.ScheduleJob(job, trigger);
```

Переваги:

набір властивостей та функцій для “Enterprise” систем;

є порт Java планувальника під .NET платформу;

підтримка контейнеризації при розгортуванні та розробці;

підтримує роботу в розподілених системах, але має ряд обмежень щодо архітектури таких систем.

Недоліки:

обмеження в архітектурі розподілених систем, що підтримуються;

відсутність широкого набору засобів контролю за алгоритмом планування;

більш складний API налаштування та журналювання завдань.

2.2.4. FluentScheduler

Простий та легковісний планувальник задач для платформи Windows. Має дуже простий інтерфейс взаємодії з API, що є зручним для програміста, як кінцевого користувача бібліотекою FluentScheduler. Приклад роботи з C# API FluentScheduler:

```
using FluentScheduler;
public class MyRegistry : Registry
{
    public MyRegistry()
    {
        Schedule<MyJob>().ToRunNow().AndEvery(2).Seconds();
        Schedule<MyJob>().ToRunOnceIn(5).Seconds();
        Schedule(() => Console.WriteLine("It's 9:15 PM
now.")).ToRunEvery(1).Days().At(21, 15);

Schedule<MyComplexJob>().ToRunNow().AndEvery(1).Months().OnThe
First(DayOfWeek.Monday).At(3, 0);

Schedule(() => new MyComplexJob("Foo",
DateTime.Now)).ToRunNow().AndEvery(2).Seconds();
Schedule<MyJob>().AndThen<MyOtherJob>().ToRunNow().AndEvery(5)
.Minutes();}}
```

Переваги:

простий API взаємодії з кінцевим застосуванням;

open-source бібліотека під .NET платформу;

підтримка контейнеризації при розгортуванні та розробці.

Недоліки:

не підтримує системи з розподіленою архітектурою;

відсутність широкого набору засобів контролю за алгоритмом планування.

ВИСНОВКИ ДО РОЗДІЛУ

Досліджено готові реалізації наявних бібліотек для планування задач в системах різного рівня та на різних платформах виконання. Представлені найпопулярніші існуючі бібліотеки планувальників задач, які працюють на різних архітектурах та операційних системах. Кожна з реалізацій має ряд переваг та недоліків, проаналізувавши які можна визначити оптимальні вимоги до функціоналу планувальника для задоволення основних потреб та досягнення максимальної швидкості виконання задач та застосування бібліотеки у розподілених Enterprise системах. Розглянемо основні недоліки готових рішень:

1. підтримка лише Windows платформи. Майже всі з розглянутих наявних бібліотек для планування задач підтримують лише Windows платформу у своїй роботі. Таким чином Enterprise системи, що розробляються як cross-platform системи не можуть використовувати розглянуті варіанти бібліотек при розробці модулів планування задач;
2. відсутність підтримки контейнеризації при використанні модуля планування у системі. Даний недолік відноситься до категорії архітектурних недоліків. При розробці Enterprise систем, що мають, наприклад, мікросервісну архітектуру, використання розглянутих бібліотек в більшості неможливо. Оскільки майже всі розглянуті бібліотеки не підтримують розгортання і використання у вигляді Docker контейнера;
3. відсутність контролю зручних засобів контролю задач, процесу виконання задач на планувальнику та вибору алгоритмів планування і конфігурації планувальника. Даний недолік повністю відноситься до бібліотек планування як до продуктів. Майже всі з розглянутих бібліотек не мають зручних засобів керування задачами, наприклад у вигляді підсистеми, яку можна інтегрувати з

продуктом, що розробляється і використовує бібліотеку планувальника.

Розглянемо основні вимоги до функціоналу додатку:

1. підтримка будь якої платформи для бібліотеки планувальника. Необхідно забезпечити сумісність бібліотеки планувальника з системою, що працює на будь якій платформі та ОС;
2. наявність зручного API інтеграції з кінцевою системою, що використовує, бібліотеку планувальника як кінцевий продукт;
3. забезпечення можливості розгортання модуля планувальника у якості docker-контейнера. Дуже важливою вимогою до бібліотеки планувальника, як до продукта, є необхідність забезпечення роботи бібліотеки у якості docker контейнера, для забезпечення можливості незалежного розгортування та підтримки планувальника, я к модуля системи.

Отже, для усунення зазначених недоліків, потрібно розробити додаток із визначеними вище вимогами. Розглянемо засоби розробки додатку. Додаток буде написаний на мові програмування C# на платформі .NET Core, що дозволить розробити cross-platform бібліотеку. Таким чином в якості основної платформи розробки серверної частини продукту обрано .NET Core. В якості бази даних продукту буде використовуватись cross-platform продукт PostgreSQL. В якості середовища розгортування продукту за замовчуванням, буде використовуватись docker контейнери та kubernetes в якості фреймворка оркестрації контейнерів. В якості основної технології frontend розробки було обрано ReactJS, оскільки це дозволить архітектурно правильно розробити системи модулів управління плануванням задач.

3 РОЗРОБКА АЛГОРИТМУ РОЗПОДІЛУ ЗАДАЧ В ПЛАНУВАЛЬНИКУ

3.1 Методи планування та управління задачами

3.1.1 Rate-monotonic

Метод призначає статичні пріоритети задачам ґрунтуючись на їх періодах. У цьому методі пріоритети визначаються наступним чином: задача з найменшим періодом отримує найвищий пріоритет.

Ця схема є оптимальною серед усіх статичних алгоритмів. Під оптимальним розуміється те, що якщо безліч задач може бути спланована будь-яким іншим статичним алгоритмом, заснованому на пріоритетах, то воно також може бути спланована і цим методом.

Вихідний RM підхід має ряд обмежень:

всі задачі повинні бути незалежні один від одного, тобто між ними немає ні взаємодії, ні загальних ресурсів;

всі задачі повинні бути періодичними;

всі задачі можуть бути припинені іншими задачами з більш високими пріоритетами. Однак жодна задача не може блокуватися, очікуючи зовнішньої події;

час виконання постійний;

для задач визначено час виконання в гіршому випадку;

всі задачі мають крайній термін, еквівалентний їх періоду.

Проведено велику кількість досліджень для розширення цих методів. В результаті цих робіт були зняті або ослаблені обмеження, що накладаються на завдання в вихідної моделі.

Так в протоколі пріоритетних кордонів (Priority Ceiling Protocol) і деяких інших схожих (Stack Resource Protocol) вдалося позбутися від обмеження на взаємодію задач. Також запропоновано багато методів приведення неперіодичних завдань до періодичних [6].

3.1.2 Deadline Monotonic

Метод може бути використаний для планування завдань, які мають крайні терміни менше або дорівнюють періодам. Він послаблює обмеження на величину крайнього терміну в схемі планування RM. У цьому випадку пріоритет, призначений завданню, обернено пропорційна величині її крайнього терміну, тобто завдання з найкоротшим крайнім терміном має найвищий пріоритет незалежно від її періоду. Якщо два завдання мають однакові крайні терміни, то вони отримують пріоритети в довільному порядку відносно один одного. Метод може обслуговувати як періодичні, так і спорадичні завдання.

Такий метод розстановки пріоритетів буде оптимальним, якщо виконуються наступні умови:

простір задач — фіксований простір жорстких задач;

задачі періодичні чи спорадичні;

задачі мають певний (відомий) час виконання у гіршому випадку;

для задач визначено критичний момент, тобто час виконання у гіршому випадку.

Оптимальність тут також означає, що якщо будь-який планувальник з фіксованими пріоритетами може спланувати простір задач, які мають крайні терміни менше або дорівнюють періоду, і виконані відповідні обмеження, то і цей планувальник теж може [7].

3.1.3 Метод фонового виконання

Найпростіший підхід , це обробляти аперіодичні задачі у фоновому режимі і запускати їх тільки тоді, коли процесор не зайнятий виконанням будь-якої з періодичних задач.

3.1.4 Метод опитування

Метод використовує окрему періодичну задачу з високим пріоритетом для підтримки виконання аперіодичних задач. Обидва ці методи неефективні, коли час відповіді аперіодичної задачі важливо.

3.1.5 Алгоритм невідкладного сервера

Це також підхід збереження пропускної здатності. Він також використовує періодичний сервер, який має найвищий пріоритет, але не обов'язково найкоротший період. Сервер припиняється, якщо не залишилося жодної аперіодичної задачі, і активізується негайно при прибутті аперіодичної задачі.

3.1.6 Алгоритм last chance

Цей алгоритм є глобально-оптимальним в тому сенсі, що забезпечує мінімальний час відповіді для аперіодичних задач (за умови виконання всіх крайніх термінів періодичних задач) серед всіх можливих методів планування періодичних і аперіодичних задач.

Планування полягає в тому, що якщо ще залишається аперіодична задача, яка повинна бути виконана, наступна періодична задача не буде запущена до самого останнього можливого моменту, званого часом повідомлення, коли ще зберігається гарантія виконання її крайнього терміну (також як і всіх інших періодичних задач) [8].

Цей метод гарантує своєчасність виконання періодичних задач і максимізує реакцію аперіодичних задач.

При використанні цього методу спочатку застосовується будь-який алгоритм з фіксованими пріоритетами для планування періодичних задач до початку роботи системи. Всі періодичні задачі мають більш високі пріоритети, ніж аперіодичні.

3.1.7 EDF

У методі EDF пріоритети задач призначаються виходячи з їх крайніх термінів на поточний момент. У цьому випадку задача з найближчим крайнім терміном отримує найвищий пріоритет. Цей метод також є оптимальним в тому сенсі, що якщо можна знайти здійсненний розклад для даного простору задач з фіксованими пріоритетами, то завжди можна знайти здійсненний розклад і з використанням цього методу. Однак він є оптимальним лише за недовантаження системи, але в умовах перевантаження веде себе досить погано.

Даний метод часто вважається небезпечним через те, що за умови перевантаження системи він може показати небажану поведінку. Однак під час роботи жорстких систем реального часу перевантажень не повинно виникати, тому що невиконання крайнього терміну задачі може привести до серйозних наслідків. Тому для таких систем необхідно проводити апріорний доказ того, що коли у всіх задач в системі одночасно виникнуть потреби в системних ресурсах, то все їх обмеження за часом і раніше будуть виконані [9].

3.1.8 Сервер, що допускає затримку і алгоритм обміну пріоритетами

Ці методи зберігають доступними ресурси системи, спочатку виділені для аперіодичних задач.

Ці методи покращують середній час відповіді системи і відрізняються один від одного способом збереження пропускної здатності. PE алгоритм роздає час виконання, виділений для роботи високопріорітетного періодичного сервера, іншим періодичним задачам з меншим пріоритетом, якщо вона не потрібна для роботи аперіодичних задач.

На відміну від нього DS не дає час виконання цієї задачі, коли не залишилося жодної аперіодичної задачі. Замість цього він зберігає цей високопріорітетний час виконання або поки не прибуде аперіодична задача, або поки не закінчиться період сервера. Цей метод простіше в реалізації, але гірше у виконанні [10].

3.2 Математична модель задачі

Є однорідна розподілена обчислювальна система, що складається з N елементарних машин (ЕМ), $N \in \mathbb{N}$ та набір з j з L незалежних задач, де

$$J = \{J_i\}, j = \overline{1, L}, L \in \mathbb{N}.$$

Кожна задача є запитом на виділення ресурсів обчислювальної системи для виконання паралельної програми і характеризується наступними параметрами: ранг і час виконання $P_j = \{p_j^k\}$, де

$$k = \overline{1, q_j}, q_j \in \mathbb{N}$$

q - кількість значень параметрів, що описують j-у програму, Кожен елемент вектора $p_j^k = (r_j^k, t_j^k)$, задає вимоги на виділення для виконання програми $rr_j^k \in E_1^N = \{a \in \mathbb{N} \mid 1 \le a \le N\}$, елементарних машин на $t_j^k \in \mathbb{N}$ одиниць часу.

Потрібно побудувати розклад виконання програм на обчислювальній системі:

$$S = \langle k_j, s_j, I_j \rangle, j = \overline{1, L}, \tag{3.1}$$

таке, що:

$$T(S) = \max_j \left\{ s_j + t_j^{k_j} \right\} \to min, \tag{3.2}$$

за умов обмежень:

$$k_j \in E_1^{q_j}, \tag{3.3}$$

$$s_j \in Z, s_j \ge 0, \tag{3.4}$$

$$I_j = \{i \in E_1^N\}, \left|I_j\right| = r_j^{k_j}, \tag{3.5}$$

$$\forall j \in E_1^L, \forall i_1 \in I_j, \forall i_2 \in I_j\{i_1\} \to i_1 \ne i_2 \tag{3.6}$$

$$\forall t \in \mathbb{N}, \bigcap_{j \in \Xi(t)} I_j = \emptyset \tag{3.7}$$

$$\forall t \in \mathbb{N}, \sum_{j \in \Xi(t)} \left|I_j\right| \le N, \tag{3.8}$$

де k_j - номер елемента списку (варіанти значень параметрів r_j^k і t_j^k), обраного для виконання j-й програми, s_j - час початку виконання -ої

програми, I_j - множина номерів елементарних машин, виділених для виконання j-ої програми, $\Xi\ (t) = \{j \in E_1^L \mid s_j \le t \le s_j + t_j^{k_j}$ - множина програм, що виконуються в момент часу t, T(S) - час виконання всіх програм набору.

Обмеження (3.5) визначає, що кожній програмі з урахуванням k_j має бути виділено стільки елементарних машин, скільки потрібно для її виконання. При цьому обмеження (3.6) і (3.7) відповідно вимагають, щоб гілки паралельної програми призначалися на різні елементарні машини, і в кожен момент часу елементарна машина була призначена на виконання однієї гілки тільки однієї паралельної програми. Обмеження (3.8) задає максимально можливу кількість гілок паралельних програм, виконуваних в кожен момент часу [11].

3.3 Алгоритм NFDH

Базова популяція формується за допомогою алгоритму NFDH (Next Fit Decreasing High). Цей алгоритм був обраний через простоту своєї реалізації і досить низької обчислювальної складності. істотним недоліком такого алгоритму є низька якість упаковки. Завдання з набору вибираються випадковим чином, потім сортуються по спадаючій часу рішення. Найбільш велике за часом завдання розташовується в лівому нижньому кутку смуги, тим самим ініціалізувавши перший рівень, по ширині рівний їй.

Решта завдань розташовуються знизу вгору, поки є місце на поточному рівні. Завдання, яке не помістилося на рівні, розміщується зверху, утворюючи наступний рівень, і так далі. Розмір популяції задається параметром алгоритму і залишається постійним в процесі всієї еволюції. Якщо для створення початкової популяції вибрати більш складний алгоритм, такий як BFDH (Best Fit Decreasing High), то розклад, знайдений генетичним алгоритмом буде більш оптимальним і наближеним до точного. Адже генетичний алгоритм НЕ створює нове, а шукає оптимальне з наявних рішень [12].

3.4 Подання набору масштабованих завдань

Набір масштабованих завдань це векторний масив, елементами якого є вектори, що містять різні варіанти запуску масштабованих завдань. Схематичне зображення масштабованого завдання в наборі представлено на рис. 3.1.

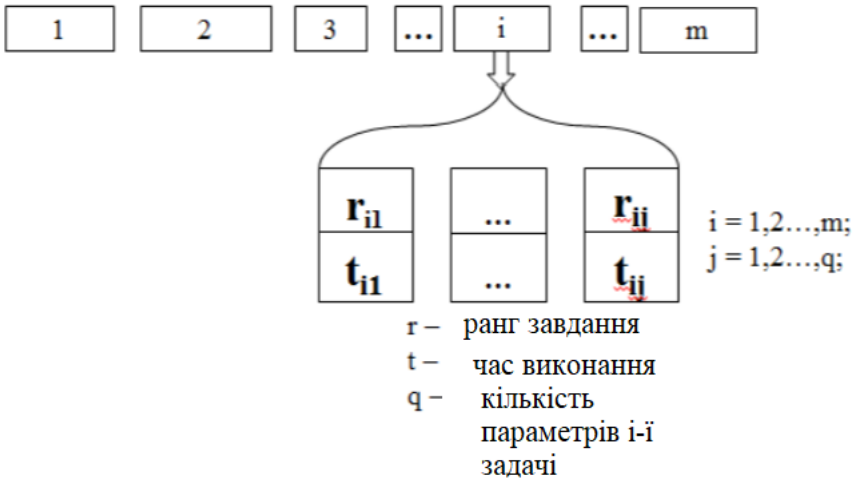


Рисунок 3.1 Масштабована задача в наборі

Кожен з елементів цих векторів в свою чергу є вектором, елементами якого є параметри запуску завдання. Масштабоване завдання може запускатися з різними параметрами:

r_n - кількість ЕМ, на яких буде виконуватися завдання,

t_n – час вирішення даного завдання.

3.5 Опис генетичного алгоритму

Одним з відомих методів вирішення завдань оптимізації та моделювання шляхом випадкового підбору є генетичні алгоритми (ГА), блок-схема якого представлена на рис. 3.2.

Генетичні, як і еволюційні алгоритми в цілому, ґрунтуються на використанні механізмів природної еволюції. Еволюція, за Дарвіном здійснюється в результаті взаємодії трьох основних факторів: мінливості, спадковості, природного відбору. мінливість грає роль основою появи нових ознак і особливостей в будові і функціях організму. Спадковість закріплює ці

ознаки. Природний відбір усуває організми, погано пристосовані до умов існування [13].

Алгоритм передбачає послідовність життєвих циклів популяції - поколінь. Кожен цикл складається з наступних операцій: селекції найбільш пристосованих особин, вибору батьківських пар, їх розмноження і мутації. Процес повторюється до тих пір, поки на протязі заданої кількості поколінь не буде з'являтися особа с кращим значенням функції пристосованості.

Рисунок 3.2 Блок-схема генетичного алгоритму

Для забезпечення роботи генетичного алгоритму були розроблені оператори селекції, схрещування особин (кросинговер) і мутації, реалізований алгоритм NFDH. Необхідно ввести такі терміни генетичного алгоритму:

ген - завдання, де вибрано k_i;

особа - допустиме розбиття задач набору при фіксованих значеннях k_i;

популяція - кілька особин з різними значеннями k_i;

покоління - життєвий цикл однієї популяції.

Цільовою функцією в даній роботі є час вирішення набору завдань. Іншими словами - час завершення вирішення останнього завдання. Існує кілька типів кросинговеру: одноточковий, двоточковий і многоточковий. Як механізм схрещування особин в даній роботі був реалізований класичний варіант одноточкового кросинговеру. Точкою розрізу особини була обрана середина. При схрещуванні відбувався обмін ділянками, обмеженими точкою розрізу.

Алгоритм оператора мутації полягав у випадковому зміну номера параметрів завдання k_i для випадкової кількості генів у особі. Ген являє собою структуру даних типу «список», що містить номер завдання, номери елементарних машин, на яких завдання має виконуватися, час запуску і час закінчення рішення. Особа – вкладений список, що включає в себе гени. Популяція представляє собою список, елементами якого є особини. Функція алгоритму проходиться по файлу з набором завдань і формує словник, в якому ключем служить номер завдання, а значення - параметри. Потім з даного словника вирогіднісно вибираються параметри, утворюючи тим самим список завдань з фіксованими параметрами. Одержаний список йде на вхід функції NFDH, яка будує розклад для початкової популяції [13].

На етапі селекції особини сортуються в порядку зростання цільової функції. Видаляється третя частина з найгіршими показниками. Тобто, на самій першій ітерації виключається ймовірність розмноження «поганих» особин. Вибір батьківських особин відбувається наступним чином: заздалегідь відсортовані особини розташовуються в порядку убування цільової функції і умовно діляться на дві рівні частини. Першою парою для схрещування вибирається перша особина з лівої частини і перша з правої, другою парою - друга з лівої і друга з правого, і так далі.

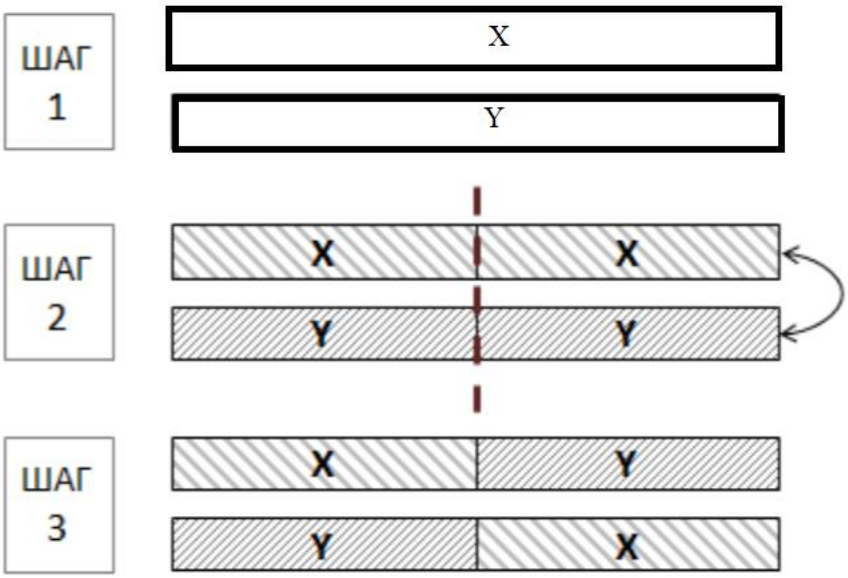


Рисунок 3.3 Схема роботи оператора кросінговер

Потім батьківські особини діляться навпіл (крок 2 на рис. 3.3), з їх частин утворюються дві нові особини (крок 3), найгірша з яких відразу вибуває, а найкраща далі бере участь в еволюції. У нову популяцію потрапляють батьківські особини і «кращі» нащадки, які продовжують брати участь в еволюції.

Функція мутації випадковим чином вибирає завдання і змінює її параметри на інші зі списку ресурсних запитів користувача. потім особина перебудовується заново за алгоритмом NFDH. Мутація може як поліпшити якість розкладу, так і погіршити. Імовірність мутації задається в програмі. Після певної кількості життєвих циклів алгоритм зупиняється. Результатом його роботи буде «найкраща» особа з останнього покоління. Критеріями зупинки може служити кількість ітерацій, або досягнення заданої різниці цільових функцій між двома поколіннями. Під другому випадку є ймовірність раннього виходу з еволюції, так як цільові функції у двох поколінь можуть бути однаковими і різниця між ними дорівнює нулю. В рамках даного алгоритму критерієм зупинки є проходження п'ятнадцяти життєвих циклів [14].

В табл. 3.1 наведено перелік функцій алгоритму планування завдань та опис вхідних даних і результатів роботи функцій.

Таблиця 3.1 Перелік функцій алгоритму

Назва функції	Призначення
`GenerateListConfig`	Проходиться по файлу з набором задач і формує словник, в якому ключем служить номер задачі, а значення параметри. Вхід: файл з набором задач; Вихід: словник з номерами задач в якості ключа і параметрами у якості значення
`NewSet`	Отримує перелік задач та формує множину цих задач, перевіряючи вхідний словник на предмет унікальності. Вхід: словник задач; Вихід: множина задач
`CreateTaskList`	З словника задач з вирогідністю вибираються параметри, утворюючи тим самим список задач з фіксованими параметрами. Вхід: множина задач; Вихід: список задач з фіксованими параметрами
`NFDH`	Будує розклад для початкової популяції. Використовує функції CreatePopulation, Selection, Crossingover у своїй роботі. Вхід: Список задач з фіксованими параметрами; Вихід: Розклад для початкової популяції
`CreatePopulation`	Створює популяцію з розкладом. Вхід: Кількість осіб у популяції; Вихід: Популяція з розкладом
`Selection`	Сортує особи за зростанням цільової функції. Вхід: Популяція; Вихід: 100 найкращих розкладів для популяції, що прийшла на вхід функції
`Crossingover`	Класичний варіант одноточкового кросинговеру. Точкою розрізу особини була обрана середина. При схрещуванні відбувався обмін ділянками, обмеженими точкою розрізу. Вхід: Покоління; Вихід: Схрещена популяція
`MainEvolution`	Виконує основну дію генетичного алгоритму, що полягає у пошуку оптимального розкладу виконання задач. Вхід: Перелік задач; Вихід: Розклад виконання задач

ВИСНОВКИ ДО РОЗДІЛУ

У даній частині роботи було проведено розгляд методів планування та управління завданнями, серед яких планування періодичних та аперіодичних завдань. Проведено аналіз даних, які можна отримати після побудування переліку завдань та вхідних параметрів для їх виконання. На основі отриманих даних описано математичну модель задачі. На основі розглянутої математичної моделі було розглянуто можливий алгоритм побудови оптимального розкладу виконання завдань у розподіленій системі.

Розроблено генетичний алгоритм побудови оптимального розкладу виконання завдань. Запропонований алгоритм полягає у комбінації алгоритму NDFN та функцій мутації та схрещення.

Функція мутації випадковим чином вибирає завдання і змінює її параметри на інші зі списку ресурсних запитів користувача. потім особина перебудовується заново за алгоритмом NFDH. Мутація може як поліпшити якість розкладу, так і погіршити. Імовірність мутації задається в програмі. Після певної кількості життєвих циклів алгоритм зупиняється. Результатом його роботи буде «найкраща» особа з останнього покоління.

Таким чином розроблений алгоритм вибору оптимального планування (розкладу) вирішення задач дозволяє мінімізувати час простою завдання в черзі, та надає змогу пришвидшити процес обробки завдань системою, та більш рівномірно розподіляти завдання між вузлами розподіленої системи, а отже уникнути проблем синхронізації даних та результатів виконання задач.

4 РЕАЛІЗАЦІЯ ГЕНЕТИЧНОГО АЛГОРИТМУ У БІБЛІОТЕЦІ ПЛАНУВАЛЬНИКА ЗАВДАНЬ

4.1 Визначення вимог і завдань

Основними функціями бібліотеки планувальника є:

1. Отримання та зберігання переліку задач, які необхідно поставити на виконання за допомогою планувальника.
2. Додавання різних типів задач до планувальника (за різними часовими періодами, інтервалами за cron виразом, тощо).
3. Відміна задач.
4. Налаштування роботи планувальника в режимі веб-ферми.
5. Фіксація та відображення поточного статусу виконання задач на планувальнику.
6. API інтеграції з системами-клієнтами.

Основні вимоги до бібліотеки планувальника:

1. Підтримка роботи в режимі web-ферми.
2. Зрозумілі конфігураційні файли, для налаштування планувальника.
3. Зрозумілий API інтеграції з клієнтськими системами.
4. Охоплення широкого спектру систем-клієнтів, підтримка cross-платформеності.

4.2 Опис функціоналу бібліотеки

Бібліотека повинна надавати користувачеві(розробнику) можливість виконувати наступні дії. Постановка задач на виконання за допомогою планувальника, що поділяється на:

1. Постановка задачі у вигляді виконуючого класу з заданим інтервалом виконання.
2. Постановка задачі у вигляді виконуючого класу з заданим часом виконання.

3. Постановка задачі у вигляді виконуючого класу за заданим cron виразом.
4. Відслідковування поточного статусу виконання завдань планувальником. Необхідно відслідковувати наступні показники:

   поточний статус виконання завдання (ОЧІКУЄ, ВИКОНУЄТЬСЯ, ЗАБЛОКОВАНЕ);

   час останнього виконання задачі;

   час наступного виконання задачі (для циклічних задач);

   назву задачі, групи задач;

   тип задачі, та налаштування її виконання (повторювати при помилці, логувати помилки чи ні і т.і.).
5. Налаштування роботи планувальника у архітектурі web-ферми. Можливість створювати декілька планувальників в рамках одного app сервера, налаштовувати кількість потоків виконання для кожного такого планувальника; Налаштування кожного екземпляру планувальника, щодо алгоритму виконання завдань на ньому, що включає такі параметри як:

   кількість потоків виконання;

   провайдер та підключення до БД;

   використання планувальника в кластері;

   налаштування повторення задач при помилках;

   налаштування пріорітетів задач на планувальнику;

   налаштування активності\неактивності конкретного планувальника на конкретному app сервері.
6. Відміна задач через їх видалення з системи, або через операції зі статусом задачі в середовищі зберігання задач планувальника.

4.3 Прецеденти

Після визначення функцій, які повинна реалізовувати бібліотека, деталізуємо сценарії використання бібліотеки та побудуємо діаграму варіантів використання. На рис. 4.1 зображена діаграма прецедентів високого рівня. На ній зображені можливі дій користувача (розробника), абстраговані від деталей. Детальніша ієрархія прецедентів зображена на рис. 4.2. – 4.4. Повна діаграма прецедентів приведена у Додатку А.

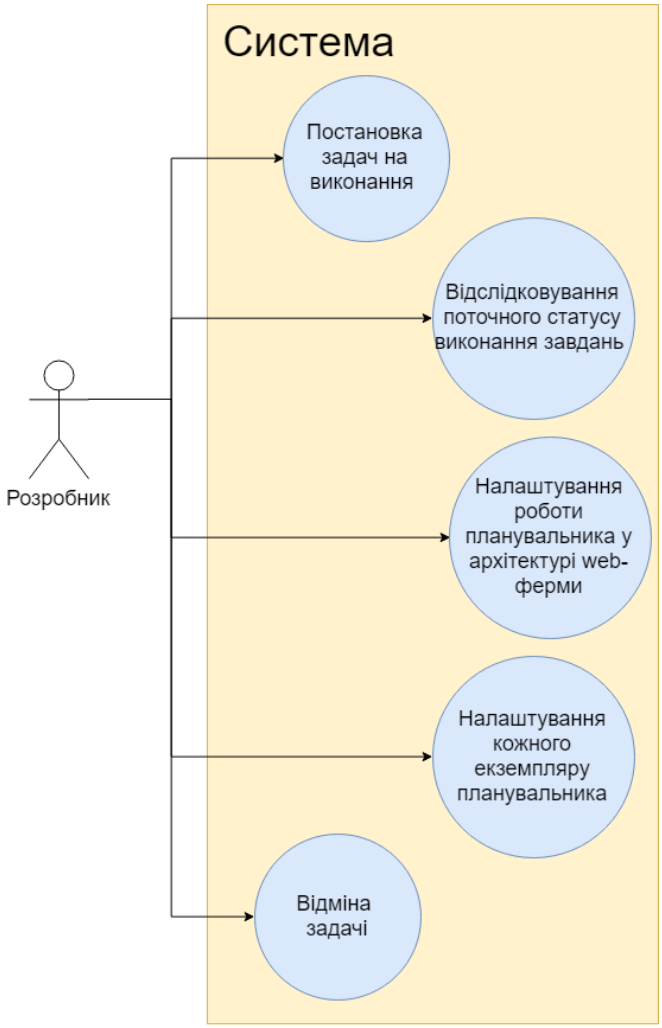


Рисунок 4.1 – Діаграма прецедентів бібліотеки

Рисунок 4.2 – Прецедент «Встановлення задач на виконання за допомогою планувальника»

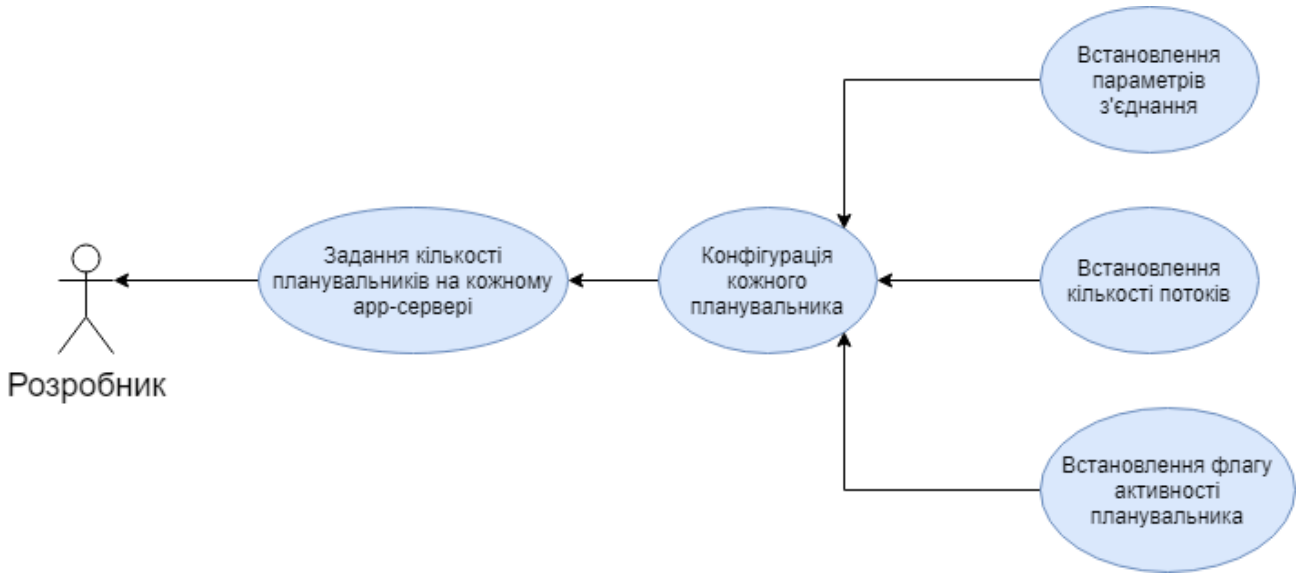


Рисунок 4.3 – Прецедент «Налаштування роботи планувальника у архітектурі web-ферми»

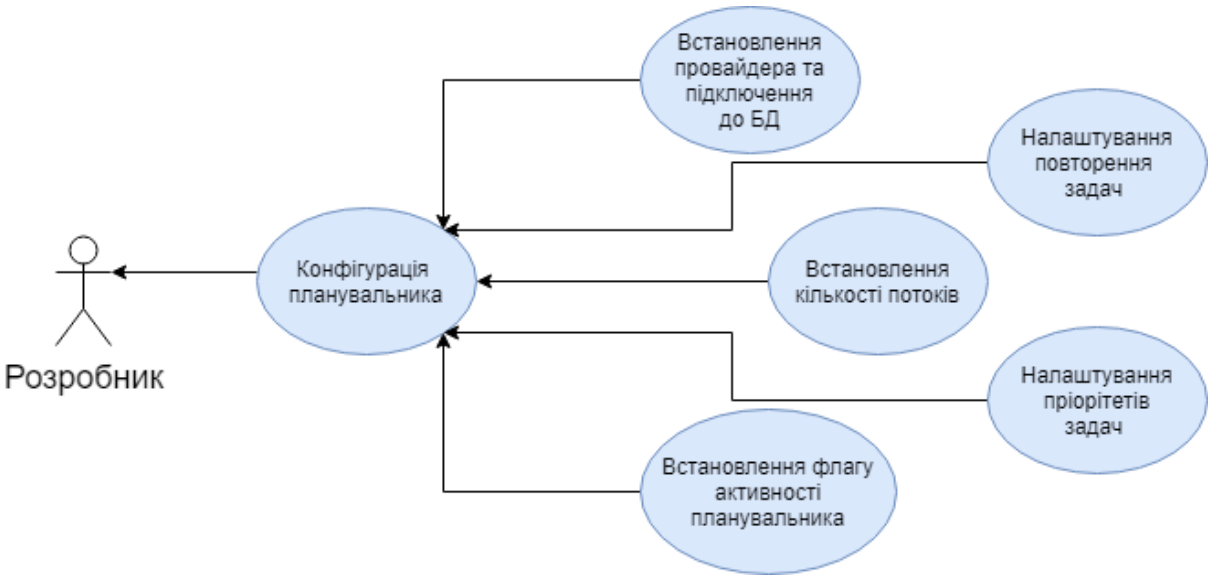


Рисунок 4.4 –Прецедент «Налаштування кожного екземпляру планувальника»

Розглянемо детальніше прецеденти:

1. Встановлення задач на виконання за допомогою планувальника.
2. Налаштування роботи планувальника у архітектурі web-ферми.
3. Налаштування кожного екземпляру планувальника.

4.3.1 Постановка задач на виконання за допомогою планувальника

Даний прецедент є одним з основних дій при використанні бібліотеки, він виконується при початку роботи з бібліотекою та у випадку, коли розробник планує встановити нову задачу на планувальник.

Цей прецедент може мати декілька сценаріїв розвитку в залежності від способу задання розкладу виконання та способу встановлення задачі на планувальник:

1. Встановлення задачі з вказаним періодом виконання за допомогою web-сервіса.
2. Встановлення задачі з вказаним часом виконання за допомогою web-сервіса.
3. Встановлення задачі з вказаним cron виразом за допомогою web-сервіса.

4. Встановлення задачі з вказаним періодом виконання за допомогою взаємодії з конфігураційною БД, файлом, тощо.
5. Встановлення задачі з вказаним часом виконання за допомогою взаємодії з конфігураційною БД, файлом, тощо.
6. Встановлення задачі з вказаним cron виразом за допомогою взаємодії з конфігураційною БД, файлом, тощо.

Детальний опис прецеденту «Постановка задач на виконання за допомогою планувальника» наведено в табл. 4.1.

Таблиця 4.1 Прецедент «Постановка задач на виконання за допомогою планувальника»

Дії користувача(розробника)	Відгук системи
Користувач (Розробник) виконує кодування задачі у вигляді класу	Система надає IntelliSense підтримку розробки та публічний API для корректності постановки задачі на планувальник
Користувач (Розробник) обирає спосіб задання розкладу виконання задачі	В залежності від обраного способу задання розкладу виконання задачі система дозволяє: 1) Обрати період виконання (рік, місяць, день, година, хвилина, секунда); 2) Обрати інтервал виконання (ціле або дробове число); 3) Ввести значення cron виразу, за яким буде виконуватись завдання
Користувач (Розробник) обирає спосіб встановлення задачі на планувальник	В залежності від обраного способу встановлення задачі на планувальник система: 1) У випадку якщо обрано web-сервіс генерує шаблон сервісу для встановлення задачі на планувальник; 2) У випадку якщо обрано встановлення задачі за допомогою БД система надає користувачу доступ на системні таблиці де містяться наявні задачі та налаштування виконання цих задач на планувальнику

4.3.2 Налаштування роботи планувальника у архітектурі web-ферми

Даний прецедент є одним з основних дій у бібліотеці, він виконується після завантаження та підключення бібліотеки до системи клієнта за допомогою інтеграції. Сценарій може мати декілька варіантів розвитку:

1. Користувач налаштовує планувальник для використання у системі, що не має розподіленої архітектури. Опис даного варіанту використання наведений у табл. 4.2.
2. Користувач налаштовує планувальник для використання у системі, з розподіленою архітектурою. Опис даного варіанту використання наведений у табл. 4.3.

Таблиця 4.2 Прецедент «Постановка задач на виконання за допомогою планувальника». Випадок налаштування з відсутністю web-ферми

Дії користувача	Відгук системи
Користувач переходить у вікно конфігурації планувальника	Система надає користувачу доступ до конфігураційних файлів планувальника з можливістю налаштування планувальника на кожному з app-серверів
У вікні конфігурації користувач вказує параметри планувальника для з'єднання з сховищем флаг його активності, кількість потоків	Система генерує config-файл з усіма вказаними параметрами планувальника, та інтегрує його в наявний конфіг системи-клієнта, або ж застосовує у якості окремого config-файла на app сервері, де розміщується система-клієнт.

Таблиця 4.3 Прецедент «Постановка задач на виконання за допомогою планувальника». Випадок налаштування з web-фермою

Дії користувача	Відгук системи
Користувач переходить у вікно конфігурації планувальника	Система надає користувачу доступ до конфігураційних файлів планувальника з можливістю налаштування планувальника на кожному з app-серверів
У вікні конфігурації користувач вказує параметри планувальника флаг його активності, кількість потоків	Система генерує config-файл з усіма вказаними параметрами планувальника
Користувач зберігає параметри планувальника	Система інтегрує config планувальника в наявний конфіг системи-клієнта, на кожному app-сервері або ж застосовує у якості окремого config-файла на app серверах, де розміщується система-клієнт. Далі система відкриває вікно редагування параметрів виконання задач планувальником на кожному з app-серверів.
Користувач змінює параметри планувальника для конкретного app-сервера	Система інтегрує config планувальника в наявний конфіг системи-клієнта, на вказаному app-сервері або ж застосовує у якості окремого config-файла на вказаному app сервері, де розміщується система-клієнт.

4.3.3 Налаштування кожного екземпляру планувальника

Даний прецедент використовується тоді, коли користувач намагається виконати налаштування окремого екземпляру планувальника на будь якому з app-серверів системи клієнта. Детальний опис прецеденту надано в табл. 4.4

Таблиця 4.4 Прецедент «Налаштування кожного екземпляру планувальника»

Дії користувача	Відгук системи
Користувач переходить у вікно конфігурації планувальника	Система надає користувачу доступ до конфігураційних файлів планувальника з можливістю налаштування планувальника на кожному з app-серверів
Користувач змінює параметри планувальника для конкретного app-сервера	Система інтегрує config планувальника в наявний конфіг системи-клієнта, на вказаному app-сервері або ж застосовує у якості окремого config-файла на вказаному app сервері, де розміщується система-клієнт.

4.4 Архітектура планувальника

4.4.1 Монолітна та мікро-сервісна архітектура

Мікро-сервіси багато в чому більше відносяться до технічних процесів, пов'язаних з пакуванням і експлуатацією, ніж до самої архітектури системи. Визначення кордонів для компонентів залишається однією з головних складнощів в інженерних системах.

Незалежно від розміру ваших сервісів, чи знаходяться вони в Docker-контейнерах чи ні завжди потрібно добре подумати про те, як зібрати систему воєдино. Немає правильних відповідей, але є безліч варіантів.

Порівняємо монолітну архітектуру і мікро-сервісний підхід. Розглянемо приклад можливої реалізації гіпотетичної платформи шаринга відео: спочатку в монолітному поданні (один великий блок), а потім в мікро-сервісному рис. 4.5.

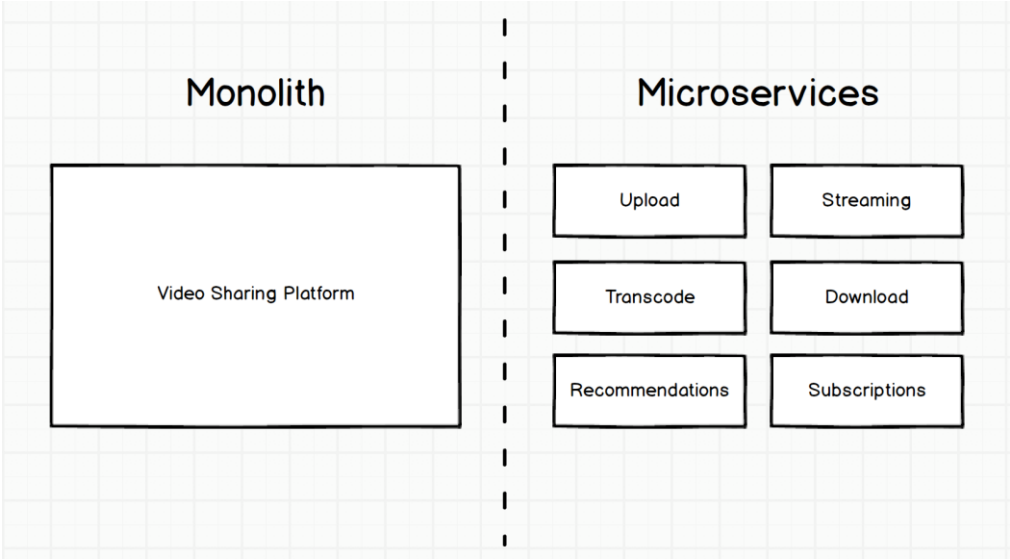


Рисунок 4.5 Порівняння монолітної та мікро-сервісної архітектур

Різниця між цими двома системами полягає в тому, що перша, це єдиний великий блок, тобто моноліт. Друга, це набір з маленьких специфічних сервісів. У кожного сервісу своя конкретна роль. Коли схема представлена на такому рівні деталізації, легко побачити її привабливість. Тут цілий набір з потенційних плюсів:

Незалежна розробка. Маленькі незалежні компоненти можуть створюватися маленькими незалежними командами. Група може працювати над змінами в сервісі Upload, не зачіпаючи сервіс Transcode і навіть не знаючи про нього. Обсяг часу, необхідного для вивчення компонента, значно знижується, і розробляти нові функції стає простіше.

Незалежне розгортання. Кожен окремий компонент можна розгортати незалежно. Це дозволяє випускати нові фічі (нову функціональність) швидко і з меншими ризиками. Виправлення або фичи для компонента Streaming можна розгортати без необхідності в розгортуванні інших компонентів [15].

Незалежна масштабованість. Кожен компонент можна масштабувати незалежно від іншого. Під час підвищеного попиту з боку користувачів, коли виходять нові передачі, компонент Download можна масштабувати для збільшення навантаження без необхідності в масштабуванні кожного компонента, що робить масштабування гнучким і знижує витрати.

Можливість повторного використання. Компоненти реалізують свою маленьку конкретну функцію. Це означає, що їх простіше адаптувати для використання в інших системах, сервісах або продуктах. Компонент Transcode може бути використаний іншими підрозділами бізнесу або навіть перетворений в новий бізнес, що пропонує послуги транскодування іншої аудиторії [16].

Можливість застосовувати різні технології. Якщо у вас є спеціалізація в конкретних сервісах, наприклад для GPU в високонавантажених обчисленнях, ви можете написати цей сервіс на C ++ або бібліотеці, яка працює безпосередньо в GPU. Якщо інші сервіси .NET вам не підходять, ви можете написати front-end на Node.js і т.д. У вас з'являється можливість вибирати різні технології. Але це не означає, що ви повинні будувати у себе «зоопарк». Стандартизація повинна бути, але якщо є достатньо підстав поміняти технологію, ви можете це зробити.

На такому рівні деталізації переваги мікро-сервісної моделі над монолітною здаються очевидними. Однак, є ряд недоліків такого підходу:

Зусилля, які потрібно витрачати кілька разів для вирішення одних і тих же завдань. У кожного сервісу з'являється свій CI, своє незалежне тестування. Існує безліч завдань, які в світі моноліту робляться один раз, і про них забувають, тоді як в світі мікро-сервісів до цих завдань потрібно постійно повертатися. Наприклад, якщо ви прийняли рішення змінити підхід до розгортання рішення, то його доведеться міняти для кожного з мікро-сервісів.

Зростає складність розробки, експлуатації, DevOps. Це дійсно так, тому що мікро-сервісна архітектура насправді складніша, ніж моноліт. У моноліті у вас все пов'язано: ви скомпілювали і переконалися, що одна частина підсистеми, на рівні контрактів, взаємодіє з іншою частиною підсистеми. У мікро-сервісній архітектурі доводиться перевіряти інтеграційними тестами, що контракти дотримані [17].

Збільшення операційних витрат. У вас може бути десяток, а то й сотня мікро-сервісів. З кожним із цих мікро-сервісів потрібно підтримувати свої процеси CI, свої процеси розгортання. За кожним із сервісів потрібно стежити незалежно: сервіс може мати обмеження по пам'яті, зв'язок з іншим сервісом

може обірватися і автоматично не відновитися, можуть бути витоки пам'яті, блокування і т.д. Набагато ускладнюється процес супроводу вашого рішення: замість одного-двох або трьох процесів доводиться стежити за сотнями.

Цілісність контрактів і даних. Кожен мікро-сервіс працює зі своїм сховищем, дані не інтегровані між собою: немає зовнішнього ключа (foreign key), і, відповідно, набагато складніше зберігати цілісність даних [18].

4.4.2 Загальна архітектура планувальника

На рис. 4.6 можна розглянути загальну схему представлення архітектури планувальника. Бачимо, що всі необхідні складові планувальника, можуть знаходитися у хмарі, а саме: сервер планувальника, сховище джобів планувальника. Також в тій же хмарі, може знаходитися і on-site база даних, та клієнт планувальника.

App-сервери взаємодіють з розміщеними в хмарі складовими планувальника і надають інформацію клієнтам, що взаємодіють з app серверами через балансувальник навантаження. Перш за все, сервер містить свій пул потоків, реалізований через Task Parallel Library. За допомогою пулу потоків отриманні завдання від сховища джобів планувальника оброблюються планувальником у фоновому режимі. Отже після того як клієнт планувальника створює задачу, вона поміщується у сховище джобів планувальника, далі сервер планувальника отримує задачу зі сховища, застосовує пул потоків для обробки завдання у фоновому режимі та віддає результат app-серверам.

4.4.3 Схема мікро-сервісів планувальника

В якості результату аналізу порівняння монолітної та мікро-сервісної архітектури слід зазначити, що всі наявні переваги мікро-сервісної архітектури можна отримати під час проектування та реалізації архітектури бібліотеки планувальника. Таким чином було вирішено використати мікро-сервісний підхід під час проектування та реалізації архітектури бібліотеки планувальника. На рис. 4.7 зображена схема мікро-сервісів бібліотеки планувальника.

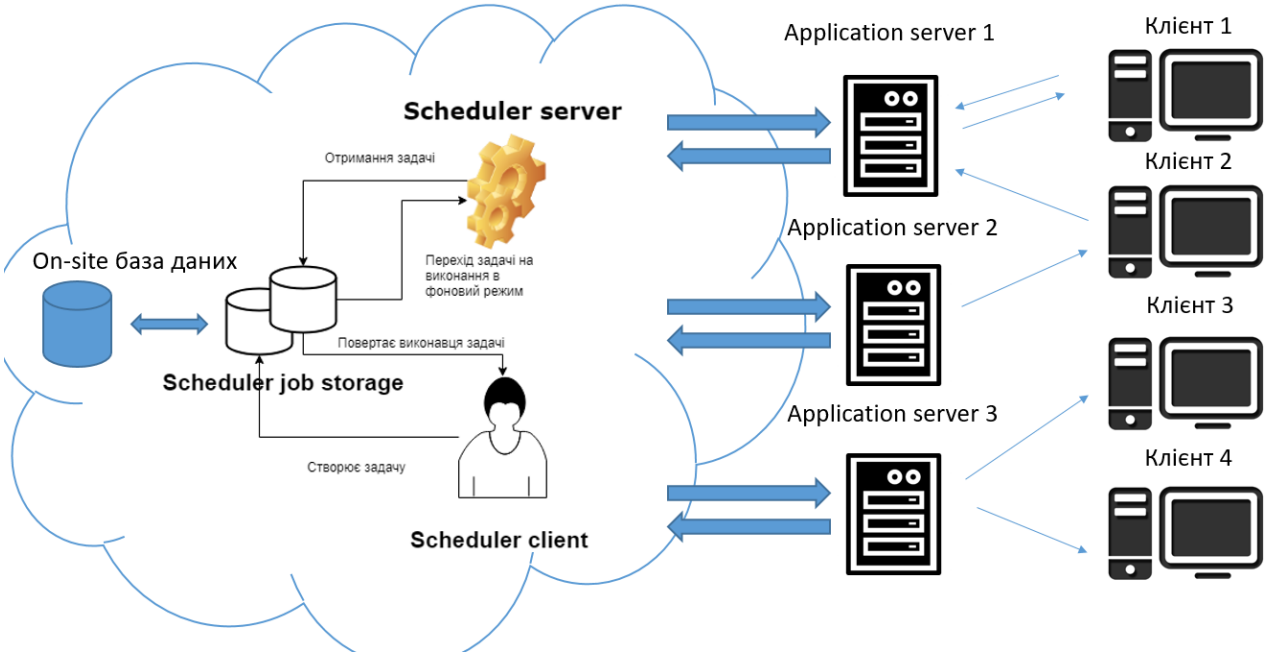


Рисунок 4.6 Загальна схема представлення архітектури планувальника

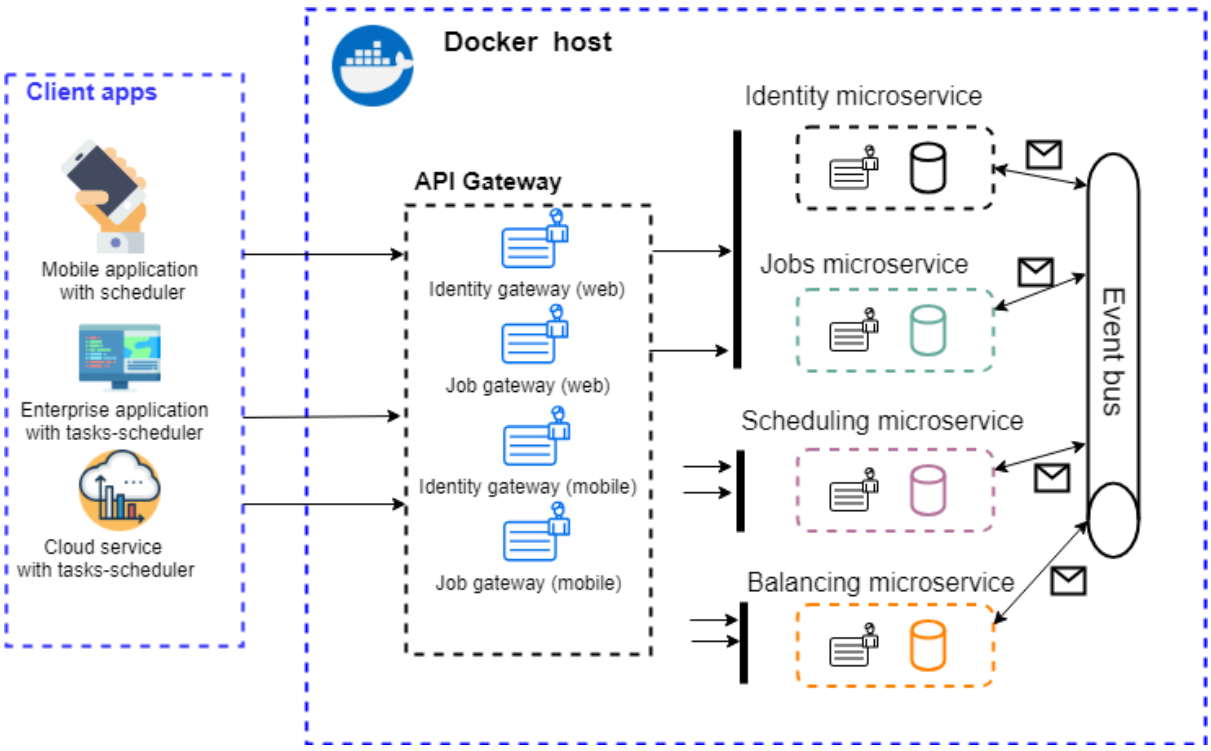


Рисунок 4.7 Схема мікро-сервісів бібліотеки планувальника

Мікро-сервісна архітектура бібліотеки має наступні загальні блоки:

1. Client apps – блок клієнтських застосувань, що використовують бібліотеку планувальника.
2. Docker host – блок Docker контейнерів, що в загальному плані складають набір мікро-сервісів планувальника, та оркеструються kubernetes.

Блок Docker контейнерів містить наступні блоки:

1. Блок API шлюзів, що є допоміжним блоком у взаємодії клієнтських застосувань з мікро-сервісами.
2. Блок мікро-сервісів, які безпосередньо виконують всю необхідну роботу бібліотеки з планування, балансування, зберігання джобів та авторизації клієнтів у планувальнику.
3. Event bus – шина повідомлень за допомогою якої організовується спілкування між мікро-сервісами.

Табл. 4.5 містить детальний опис вхідних та вихідних даних, а також основні функції мікро-сервісів бібліотеки планувальника. Розглянемо детально вхідні та вихідні дані а також основні функції кожного з мікро-сервісів:

identity;

jobs;

scheduling;

balancing;

Таблиця 4.5 Мікро-сервіси бібліотеки планувальника

Мікросервіс	Вхідні дані	Результат дії	Основні функції
`Identity`	Конфігурація підключення до джерела даних, системний користувач, що запустив планувальник	Результат авторизації бібліотеки в підсистемі джерела даних та користувача в бібліотеці планувальника.	Авторизація бібліотеки в підсистемі джерела даних та користувача в бібліотеці планувальника.

Таблиця 4.5 Мікро-сервіси бібліотеки планувальника

`Scheduling`	Колекція системних джобів, готових до планування та балансування в системі.	Розклад виконання завдань з урахуванням пріоритетів задач та прогнозованого часу їх виконання на планувальнику	Застосування генетичного алгоритму створення розкладу виконання задач на планувальнику з урахуванням розподіленості системи.
`Balancing`	Колекція системних джобів, готових до планування та балансування в системі.	Колекція відповідностей між джобом, що необхідно виконати та app-сервером на якому буде виконуватися завдання	Застосування алгоритмів балансування у розподілених системах, для забезпечення рівномірного навантаження на кожній з app-серверів на яких працює планувальник.

Розглянувши детально кожний з наведених в таблиці мікро-сервісів, ми бачимо, що майже все сервіси використовують певний «стан» системи і є так званими statefull сервісами. Виходячи з цього при розробці системи постає проблема зберігання стану та його синхронізації і розподілення між декількома мікро-сервісами.

4.5 Архітектурні шаблони роботи з базою даних у мікро-сервісах

Розглянемо 3 основних підходи до вирішення даної проблеми. Існує також безліч супутніх шаблонів.

4.5.1 Shared database

Система має одну базу даних, яка використовується декількома службами. Кожна служба вільно звертається до даних, що належать іншим службам, використовуючи локальні транзакції ACID [19].

Переваги цієї моделі:

розробник використовує знайомі і прості транзакції ACID для забезпечення узгодженості даних;

єдина база даних простіше для роботи;

Недоліками цього шаблону є:

зв'язок часу розробки - розробник, який працює, наприклад, з OrderService, потрібно скоординувати зміни схеми з розробниками інших служб, які звертаються до тих же таблиць. Цей зв'язок і додаткова координація сповільняють розвиток;

зв'язок часу виконання - оскільки всі служби отримують доступ до однієї і тієї ж бази даних, вони можуть потенційно заважати один одному. Наприклад, якщо тривала транзакція CustomerService містить блокування в таблиці ORDER, тоді OrderService буде заблокований;

єдина база даних може не задовольняти вимогам зберігання і доступу до даних всіх служб.

4.5.2 Database per service

Необхідні умови: Сервіси повинні бути слабо пов'язані, щоб їх можна було розробляти, розгортати і масштабувати незалежно [19].

Деякі бізнес-транзакції повинні забезпечувати дотримання варіантів, що охоплюють кілька сервісів. Наприклад, для створення нового замовлення на розміщення, необхідно перевірити, що нове замовлення не буде перевищувати кредитний ліміт клієнта. Інші бізнес-транзакції повинні оновлювати дані, що належать кільком службам.

Деякі бізнес-транзакції повинні запитувати дані, що належать кільком службам. Наприклад, використання View Available Credit має запитувати у Клієнта, щоб знайти creditLimit і Orders, щоб обчислити загальну суму відкритих замовлень. Деякі запити повинні повертати агреговані дані, що належать кільком службам. Наприклад, пошук клієнтів в певному регіоні і їх недавні замовлення вимагають з'єднання між клієнтами і замовленнями.

Бази даних іноді повинні бути такими, що реплікуються і готові до масштабування. (ScaleCube). Різні служби мають різні вимоги до сховища даних. (SQL NoSQL). База даних сервісу фактично є частиною реалізації цього сервісу. Доступ до даних безпосередньо не можливий іншими сервісами.

Існує кілька способів збереження конфіденційних даних сервісу. Вам не потрібно надавати сервер бази даних для кожного сервісу. Наприклад якщо використовується реляційна база даних, можливі наступні варіанти:

private-tables-per-service - кожна служба має набір таблиць, до яких повинна бути доступна ця служба;

schema-per-service - кожна служба має схему бази даних, яка є приватною для цієї служби;

database-server-per-service - кожна служба має власний сервер бази даних.

Використання бази даних для кожної служби має такі переваги:

допомагає гарантувати, що сервіси слабо пов'язані. Зміни в базі даних однієї служби не впливають на інші служби;

кожна служба може використовувати тип бази даних, який найкраще підходить для її потреб.

Використання бази даних для кожної служби має такі недоліки:

впровадження бізнес-транзакцій, що охоплюють кілька сервісів, не є простим. Розподілених транзакцій краще уникати через теореми CAP. Більш того, багато сучасних (NoSQL) баз даних не підтримують їх. Краще рішення - використовувати шаблон Saga. Служби публікують події при оновленні даних. Інші служби підписуються на події і оновлюють свої дані у відповідь;

реалізація запитів, які об'єднують дані, які зараз знаходяться в декількох базах даних, є складним завданням.

Існують різні рішення задачі об’єднання даних, насамперед такі як:

API Composition - додаток виконує з'єднання, а не база даних. Наприклад, служба (або шлюз API) може отримати клієнта і їх замовлення, спочатку витягуючи клієнта зі служби клієнтів, а потім запитуючи службу замовлення, щоб повернути найостанніші замовлення клієнта.

Сегментація відповідальності запиту запитів (CQRS) — підтримує одне або кілька матеріалізованих уявлень, що містять дані з декількох служб.

4.6 Вибір технологій та їх обґрунтування

4.6.1 Вибір основної платформи для бібліотеки

Виходячи з вимог, що були поставлені на початку роботи по розробці бібліотеки планування завдань в розподілених системах, необхідно досягнути підтримки максимальної кількості потенціальних платформ на яких можуть буди розгорнуті клієнтські системи, що будуть використовувати бібліотеку планувальника. Також слід звернути увагу на архітектуру планувальника. При проектуванні архітектури було вирішено використовувати мікро-сервісний підхід, що є дуже важливим фактором, що впливає на вибір основної платформи для бібліотеки. Розглядалося дві можливих основних платформи розробки:

1. Java.
2. NET Core.

Програмна платформа Java - ряд програмних продуктів і специфікацій компанії Sun Microsystems, раніше незалежної компанії, а нині дочірньої компанії корпорації Oracle, які спільно надають систему для розробки прикладного програмного забезпечення та вбудовування її в будь-яке крос-платформене програмне забезпечення. Java використовується в різних комп'ютерних платформах від вбудованих пристроїв і мобільних телефонів в нижньому ціновому сегменті, до корпоративних серверів і суперкомп'ютерів у вищому ціновому сегменті. Хоча Java-аплети рідко використовуються в настільних комп'ютерах, проте вони іноді використовуються для поліпшення функціональності і підвищення безпеки.

Програмний код, написаний на Java, віртуальна машина Java виконує байт-код Java. Однак є компілятори байт-коду для інших мов програмування, таких як Ada, JavaScript, Python, і Ruby. Також є кілька нових мов програмування, розроблених для роботи з віртуальною машиною Java. Це такі мови як Scala, Clojure and Groovy. Синтаксис Java в основному запозичений з Сі і C ++, але об'єктно-орієнтовані можливості засновані на моделі, використовуваної в Smalltalk і Objective-C. В Java відсутні певні низькорівневі конструкції, такі як

покажчики, також Java має дуже просту модель пам'яті, де кожен об'єкт розташований в купі і всі змінні об'єктного типу є посиланнями. Управління пам'яттю здійснюється за допомогою інтегрованої автоматичної збірки сміття, яку виконує JVM.

13 листопада 2006 року компанія Sun Microsystems зробила велику частину своєї реалізації Java доступною відповідно до GNU General Public License (GPL), хоча деякі частини поставляються в скомпільованому вигляді через питання авторського права з кодом, на який має ліцензію (але не право власності) компанія Sun Microsystems

.NET Core та платформа ASP.NET Core представляє технологію від компанії Microsoft, призначену для створення різного роду веб-додатків: від невеликих веб-сайтів до великих веб-порталів і веб-сервісів.

З одного боку, ASP.NET Core є продовженням розвитку платформи ASP.NET. Але з іншого боку, це не просто черговий реліз. Вихід ASP.NET Core фактично означає революцію всієї платформи, її якісна зміна.

Розробка над платформою почалася ще в 2014 році. Тоді платформа умовно називалася ASP.NET vNext. У червні 2016 року вийшов перший реліз платформи. А в травні 2018 року побачила версія ASP.NET Core 2.1.

ASP.NET Core тепер повністю є opensource-фреймворком. Всі вихідні файли фреймворку доступні на GitHub.

ASP.NET Core може працювати поверх крос-платформної середовища .NET Core, яка може бути розгорнута на основних популярних операційних системах: Windows, Mac OS X, Linux. І таким чином, за допомогою ASP.NET Core ми можемо створювати крос-платформні додатки. І хоча Windows як середовище для розробки і розгортання програми досі є більш популярним, але тепер вже ми не обмежені тільки цією операційною системою. Тобто ми можемо запускати веб-додатки не тільки на ОС Windows, але і на Linux і Mac OS. А для розгортання веб-додатку можна використовувати традиційний IIS, або крос-платформний веб-сервер Kestrel.

ASP.NET Core переважно націлений на використання .NET Core, але фреймворк також може працювати і з повною версією фреймворка .NET.

Завдяки модульності фреймворка всі необхідні компоненти веб-додатки можуть завантажуватися як окремі модулі через пакетний менеджер Nuget. Крім того, на відміну від попередніх версій платформи немає необхідності використовувати бібліотеку System.Web.dll.

ASP.NET Core характеризується розширюваністю. Фреймворк побудований з набору незалежних компонентів. І ми можемо або використовувати вбудовану реалізацію цих компонентів, або розширити їх за допомогою механізму спадкування, або зовсім створити і застосовувати свої компоненти зі своїм функціоналом.

Також було спрощено управління залежностями і конфігурація проекту. Фреймворк тепер має свій легкий контейнер для впровадження залежностей, і більше немає необхідності застосовувати сторонні контейнери, такі як Autofac, Ninject. Хоча при бажанні їх також можна продовжувати використовувати.

Проаналізувавши обидві платформи та зіставивши їх переваги та недоліки з вимогами до архітектури та системи в загалому в якості основної платформи розробки було обрано .NET Core, оскільки, якщо підсумувати, то можна виділити наступні ключові відмінності ASP.NET Core [20]:

легкий і модульний конвеєр HTTP-запитів;

можливість розгортати додаток як на IIS, так і в рамках свого власного процесу;

використання платформи .NET Core і її функціональності;

поширення пакетів платформи через NuGet;

інтегрована підтримка для створення та використання пакетів NuGet;

єдиний стек веб-розробки, що поєднує Web UI і Web API;

конфігурація для спрощеного використання в хмарі;

вбудована підтримка для впровадження залежностей;

можливість розширення;

кросплатформеність: можливість розробки і розгортання додатків ASP.NET на Windows, Mac і Linux;

розвиток як open source, відкритість до змін.

4.6.2 Вибір архітектури розгортування

Оскільки архітектура планувальника основана на мікро-сервісному підході то важливим аспектом розробки є вибір архітектури розгортування бібліотеки, яка буде підтримуватись продуктом за замовчуванням. Ця ж архітектура розгортування буде використовуватись під час розробки та тестування продукту, а також при виході на prod середовище у вигляді демо-застосувань. В якості архітектури розгортування планувальника було обрано Docker та Kubernetes, як засіб контейнерного оркестрування.

Docker — програмне забезпечення для автоматизації розгортання і управління додатками в середовищі віртуалізації на рівні операційної системи. Дозволяє «упакувати» додаток з усім його оточенням і залежностями в контейнер, який може бути перенесений на будь-яку Linux-систему з підтримкою cgroups в ядрі, а також надає середовище з управління контейнерами. Спочатку використовував можливості LXC, з 2015 року застосовував власну бібліотеку, що абстрагує віртуалізаційні можливості ядра Linux - libcontainer. З появою Open Container Initiative почався перехід від монолітної до модульній архітектурі.

Таким чином, було прийнято рішення по розробці планувальника у якості набору Docker контейнерів, які буде легко розгортувати на будь якій платформі, що вирішить задачу cross-платформеності планувальника, та вирішить ряд проблем, що виникають при розробці ПЗ.

Kubernetes — відкрите програмне забезпечення для автоматизації розгортання, масштабування і управління контейнеризованих додатків. Підтримує основні технології контейнеризації, включаючи Docker, rkt, також можлива підтримка технологій апаратної віртуалізації.

Оригінальна версія була розроблена компанією Google для внутрішніх потреб, згодом система передана під управління Cloud Native Computing Foundation. Використовуються низкою великих організацій і інтернет-проектів, зокрема, інфраструктура фонду Wikimedia Foundation перенесена з самостійно розробленого програмного забезпечення для організації кластерів на Kubernetes [21].

Kubernetes буде використовуватись в якості платформи оркестрації за замовчуванням.

4.6.3 Вибір мови програмування

Для розробки додатку доцільно обрати мову програмування C#. З використанням основних нових можливостей C# 6.0 та C# 7.0. Оскільки це стандартна мова для розробки додатків на платформі .NET Core, використання якою підтримує компанія-розробник платформи. Офіційна документація для .NET Core розроблена з розрахунку використання саме цієї мови. Також у світі існує досить велика спільнота C# програмістів, що спрощує вирішення можливих проблем із реалізацією, шляхом обговорення на публічних інтернет-ресурсах.

4.6.4 Вибір допоміжних бібліотек

Для платформи .NET існує велика кількість бібліотек із відкритим кодом, які значно спрощують та пришвидшують процес створення додатків, допомагають уникнути повторюваності шаблонного коду та реалізувати зручну для розробки та підтримки готових проектів архітектуру. На жаль, враховуючи досить «свіжу» дату виходу платформи .NET Core, ще не всі популярні бібліотеки було переведено на нову платформу, але їх кількість стрімко зростає та вони доповнюються новою функціональністю, якої не було у версіях цих бібліотек для платформи .NET. У проекті використовуються наступні бібліотеки:

1. Swagger;
2. Log4Net;
3. AutoMapper;
4. Newtonsoft;
5. NpgSQL.

Swagger — це фреймворк і специфікація для визначення REST APIs в форматі, дружньому до користувача і комп'ютера (JSON або YAML). Але Swagger - це не просто специфікація. Основна його функція полягає в додаткових інструментах. Для Swagger існує величезна кількість безкоштовних утиліт (як офіційних, так і написаних сторонніми розробниками), які можуть зробити розробку більш якісною та швидкою. Ви можете встановити все це на свої власні сервера і подивитися, як це працює - наприклад, спробувати роботу з браузером документів або Swagger онлайн-редактором.

log4net — порт фреймоворка для логування log4j на платформу Microsoft .NET Framework. Первісна робота була виконана компанією Neoworks і підтримано організацією Apache Software Foundation в лютому 2004. log4net - інструмент, що допомагає програмісту отримувати лог записів для різних цілей. Фреймворк схожий на оригінальний log4j, але при цьому має переваги у вигляді нових можливостей середовища виконання .NET. Є підтримка Nested Diagnostic Context (NDC) і Mapped Diagnostic Context (MDC).

AutoMapper — це об'єктно-об'єктний маппер. Об'єктно-об'єктний мапінг працює шляхом перетворення вхідного об'єкта одного типу в вихідний об'єкт іншого типу. Що робить AutoMapper цікавим тим, що він надає деякі цікаві шляхи роботи, щоб взяти роботу з з'ясування того, який мапінг типу A для типу B. Поки тип B слідує за встановленою конвенцією AutoMapper, для мапінгу двох типів потрібна практично нульова конфігурація.

Newtonsoft — популярна бібліотека для роботи з об’єктами у форматі JSON на платформі .NET. Бібліотека дозволяє мінімізувати роботу по обробці JSON об’єктів у коді на платформі .NET.

Npgsql — це постачальник даних ADO.NET з відкритим вихідним кодом для PostgreSQL, він дозволяє програмам, написаним у C #, Visual Basic, F # організовувати доступ до сервера баз даних PostgreSQL. Бібліотека має реалізацію на C#, є безкоштовною і має відкритий вихідний код. Крім того, бібліотека містить провайдери для Entity Framework Core і Entity Framework 6.x.

4.7 Реалізація бібліотеки та її компонентів

Процес розробки бібліотеки можна умовно поділити на декілька основних підпроцесів, а саме:

реалізація Scheduling та Balancing мікро-сервісів;

реалізація і налаштування Job мікро-сервісу;

реалізація API взаємодії з планувальником для систем-клієнтів.

Для подальшого поглиблення у реалізацію бібліотеки планувальника слід розглянути верхньорівневу структуру його класів. Всі класи планувальника відповідно до їх призначення розташовано у відповідних просторах імен, назви та основне призначення яких вказано у табл. 4.6. Кожний простір імен містить групу класів, що відповідають за конкретний блок функціональності планувальника. В таблиці надано перелік всіх просторів імен та вказано назву мікро-сервісу до якого відноситься той чи інший простір.

Таблиця 4.6 Простори імен бібліотеки планувальника

Простір імен	Опис	Мікро-сервіс
`Scheduler`	Простір імен, що містить основні абстракції та класи необхідні для побудови розкладів, конвертації задач у джоби планувальника	Scheduling
`Core`	Простір імен, що містить основні класи необхідні для роботи планувальника а також пулу потоків за допомогою якого задачі виконуються планувальником	Identity Scheduling
`Impl`	Простір імен, що містить класи для роботи фабрик джобів, репозиторіїв джобів, а також класи, що зберігають інформацію про деталі кожної з задач	Scheduling
`Impl. AdoJobStore`	Простір імен, що містить класи необхідні для реалізації можливості зберігання джобів та деталей їх виконання у базі даних	Job

Таблиця 4.6 Простори імен бібліотеки планувальника

Impl. Calendar	Простір імен, що містить класи необхідні для роботи з різними календарями при запуску задач на планувальнику	Balancing
Impl. Matchers	Простір імен, що містить допоміжні класи для перевірки відповідності джобів до статусу придатного для запуску	Scheduling
Impl .Triggers	Простір імен, що містить класи тріггерів – сутностей необхідних для запуску завдань відповідно до розкладу їх виконання.	Scheduling
Listener	Простір імен, що містить абстракції та реалізації механізмів прослуховування. Ці Механізми необхідні для запуску задач за певними умовами.	Balancing
Logging	Простір імен, що містить класи необхідні для логування дій планувальника. Класи потрібні для швидкого отримання даних про поточний статус виконання задач та швидкого повідомлення про помилки під час роботи.	Scheduling
Simpl	Простір імен, що містить клас-обгортки для класів платформи .NET, що додають деякі необхідні для роботи планувальника функції до стандартної поведінки.	Scheduling
Xml	Простір імен, що містить допоміжні класи для обробки та зберігання системної інформації в xml файлах	Job

4.7.1 Scheduling та Balancing мікро-сервіси

Основна логіка роботи Scheduling та Balancing мікро-сервісів належить класам ScheduleContext та ScheduleBuilder. Розглянемо роботу зазначених вище класів більш детально.

ScheduleContext - утримує дані контексту / середовища, які можуть бути доступні джобу планувальника, коли вони виконуються. Властивості класу ScheduleContext зазначені в табл. 4.7.

Таблиця 4.7 Перелік властивостей класу ScheduleContext

Назва властивості	Опис
`Count`	Повертає кількість елементів в колекції контексту
`Dirty`	Визначає чи помічений контекст як «Змінений»
`IsEmpty`	Визначає чи є контекст порожнім
`IsFixedSize`	Визначає чи є контекст фіксованим, чи може розширюватись з додаванням нових задач
`IsReadOnly`	Визначає чи можна редагувати контекст
`Item`	Повертає задачу з контексту за ключем
`Keys`	Повертає ключі задач з якими їх було додано до контексту
`SyncRoot`	Флаг, що позначає можливість\неможливість синхронного доступу до контексту з декількох потоків
`Values`	Задачі, що знаходяться в контексті

Бачимо, що властивості класу ScheduleContext дозволяють отримати доступ до джобів, що належать даному контексту та отримати інформації про поточний стан контексту. Розглянемо поведінку та основні методи класу ScheduleContext, що зазначені в табл. 4.8.

Таблиця 4.8 Методи класу ScheduleContext

Назва метода	Опис
`Add(KeyValuePairTKey, TValue)` та його перевантаження	Додає задачу до контексту
`Clear` та його перевантаження	Очищує контекст планувальника
`ClearDirtyFlag`	Дозволяє встановити флаг «зміненості» контекста
`Clone`	Виконує клонування контексту
`Contains(KeyValuePairTKey, TValue)` та його перевантаження	Перевіряє наявність задачі у контексті

Таблиця 4.8 Методи класу ScheduleContext

`Get` та його перевантаження	Отримує задачу з контексту за ключем
`Put(TKey, TValue)` та його перевантаження	Змінює задачу у контексті за вказаним ключем
`Remove(KeyValuePairTKey, TValue)` та його перевантаження	Видаляє задачу з контексту
`TryGetValue`	Безпечно отримує задачу з контексту, з перевіркою її наявності перед операцією діставання

Таким чином за допомогою класу ScheduleContext всі задачі, що встановлюються на планувальник виконуються у єдиному контексті, яким ми можемо керувати за допомогою розглянутого класу. Розглянемо клас ScheduleBuilder та його дочірні класи. Ієрархія цих класів виконує безпосередньо визначення розкладу виконання задач планувальником та зазначена на рис. 4.8.

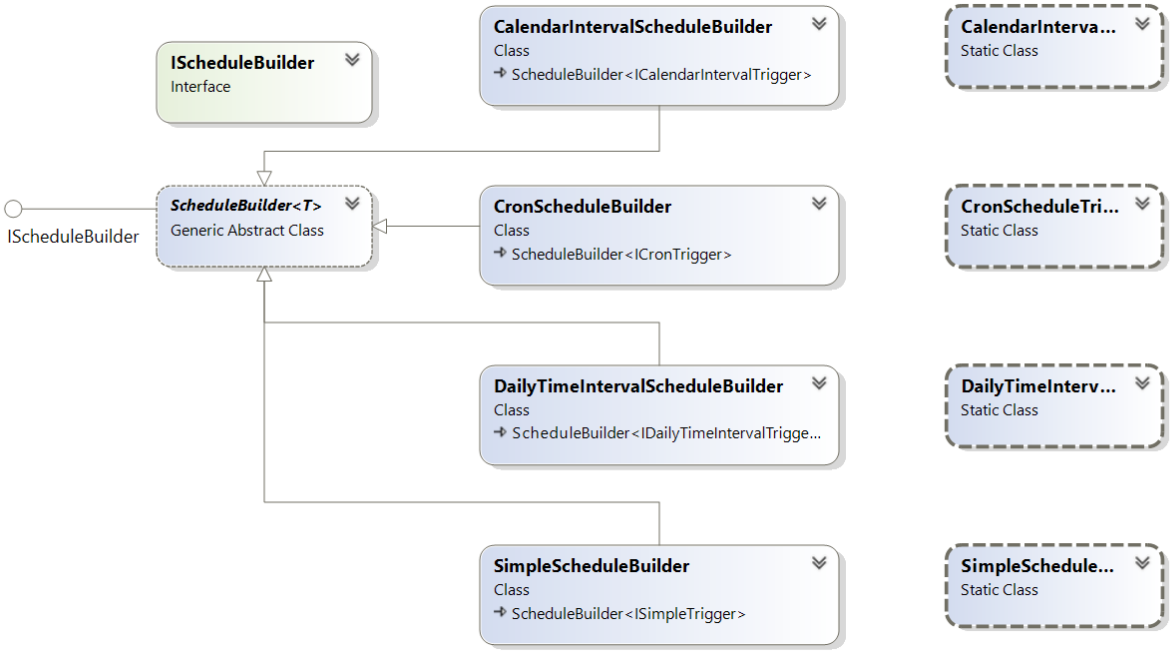


Рисунок 4.8 Ієрархія класів побудови розкладу виконання задач

Бачимо, що ієрархія класів необхідних для побудови розкладу задач різних типів, що виконуються планувальником містить наступні частини:

1. IScheduleBuilder – базовий публічний інтерфейс «будівельника» розкладу.
2. ScheduleBuilder – базовий клас «будівельника» розкладу, містить основні функції будівельника, що дозволяють виконати побудову розкладу, та можуть бути змінені дочірніми класами для забезпечення можливості виконання задач різного типу.
3. CalendarIntervalScheduleBuilder – клас побудови розкладу задач, що виконуються за календарним інтервалом.
4. CronScheduleBuilder – клас побудови розкладу задач, що виконуються за cron виразом.
5. DailyTimeIntervalScheduleBuilder – клас побудови розкладу задач, що виконуються за інтервалом впродовж доби.
6. SimpleScheduleBuilder – клас побудови розкладу задач, що виконуються за простим розкладом.

Для подальшого розгляду Schedule мікро-сервіса слід зазначити, що клас ScheduleBuilder містить єдиний public метод Build, який призначений для побудови розкладу виконання задачі і реалізується для різних типів задач у дочірніх класах ScheduleBuilder.

Balancing мікро-сервіс відповідає за балансування задач між потоками-обробниками. Даний мікро-сервіс необхідний для зменшення часу простою задач у черзі і представлений наступною групою класів:

1. ListenerBroadcastJobListener – прослуховувач пулу потоків, що направляє вільні задачі на виконання у поток пулів.
2. ListenerBroadcastSchedulerListener – прослуховувач пулу класів, що виконують розробку розкладу для виконання задачі. Прослуховує пул на предмет виявлення класів, що можуть виконати розробку розкладу для нової задачі, що поступила до планувальника.
3. ListenerBroadcastTriggerListener – прослуховувач трігерів, що надає коллекцію тріггерів готових до винонання нової задачі.

4. ListenerJobChainingJobListener– прослуховувач задач, що виконуються у ланцюгу, це такі задачі які виконуються за принципом: наступна задача виконується лише після завершення попередньої з використанням результату її виконання.

4.7.2 Job мікро-сервісу

Job мікро-сервіс відповідає за зберігання інформації про існуючі задачі у планувальнику, їх розклади та налаштування виконання. Job мікро-сервіс має надавати змогу роботи з декількома типами зберігання інформації, основними з яких є XML файли та база даних.

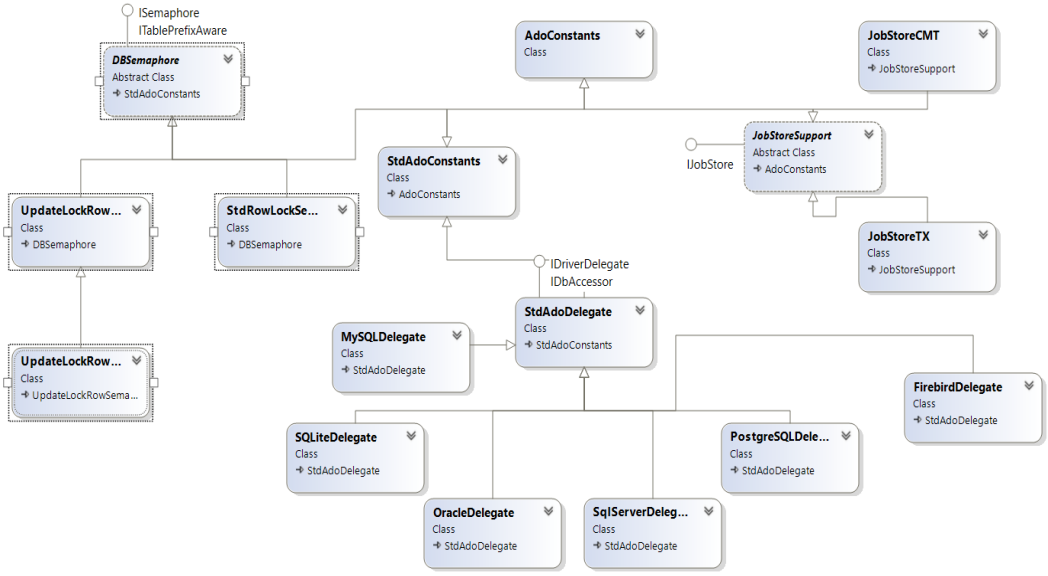


Рисунок 4.9 Ієрархія класів Job мікро-сервісу для роботи з БД

Класи мікро-сервісу, що відповідають за зберігання конфігураційної інформації та інформації часу виконання у базі даних мають ієрархію, що зображена на рис. 4.9. Класи наведені на діаграмі можна умовно розподілити на три групи:

1. Класи, що відповідають за збереження даних про задачі планувальника-JobStore та його дочірні класи.

2. Класи, що є конекторами мікро-сервісу до різних типів баз даних – StdAdoConstants та його дочірні класи для конкретних баз даних.
3. Класи синхронізації доступу до бази даних з пулу потоків планувальника – DBSemaphore та його дочірні класи.

Частина Job мікро-сервісу, що відповідає за зберігання інформації про задачі планувальника у вигляді XML файлів представлена класом XMLSchedulingDataProcessor, що містить всі необхідні властивості та функції для забезпечення роботи по зберіганню та зчитуванню конфігураційних даних та даних часу виконання задач планувальника.

4.7.3 API взаємодії з планувальником для систем-клієнтів

Реалізація API взаємодії з планувальником для систем-клієнтів розглянемо на конкретному прикладі використання розробленої бібліотеки планувальника у реальному проекті. Розглянемо ієрархію класів зображену на рис. 4.10

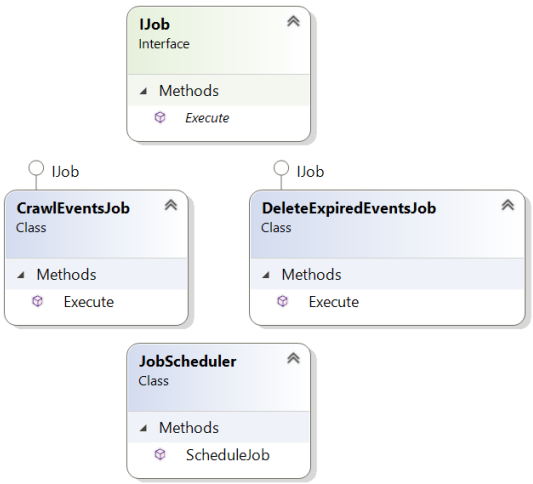


Рисунок 4.10 Приклад ієрархії класів для використання API планувальника

API взаємодії з планувальником реалізовано за допомогою класу JobScheduler, який дозволяє запланувати задачу на виконання з заданим періодом виконання або за заданим cron виразом. Клас має єдиний public метод Schedule, що дозволяє розмістити задачу на виконання планувальником.

Реалізація конкретних задач у системі відбувається шляхом наслідування від інтерфейсу IJob, який має єдиний public метод (точку входу), який має містити всю логіку, необхідну для виконання задачі, яку він описує. В нашому прикладі це дві задачі CrawlEventsJob та DeleteExpiredEventsJob. Реалізація метода Execute у класах, що реалізовують інтерфейс IJob може бути будь-якою і містити будь які операції, які розробник, користувач хоче виконати за допомогою планувальника.

Структура класів необхідних для запуску планувань наведена на рис. 4.9, вказаних класів цілком достатньо для того щоб виконати задачі: CrawlEventsJob та DeleteExpiredEventsJob за допомогою планувальника. Розглянемо приклад взаємодії з API бібліотеки, який дозволяє встановити вказані задачі на планувальник для виконання в необхідний час. Фрагмент коду, який виконує поставлену вище задачу наведено нижче. Фрагмент встановлення виконання завдання на планувальнику:

```
var crawlEventsJob = new CrawlEventsJob();
var deleteExpiredEventsJob = new DeleteExpiredEventsJob();
var updateNearestEventsJob = new UpdateNearestEventsJob();

var deleteExpiredEventsExecutionSettings = new JobExecutionSettings()
{
    DbConnectionString = dbConnectionString,
    BaseAddress = baseAddress,
    FacebookToken = facebookToken,
    JobPeriodType = JobPeriodType.Day,
    JobPeriod = 1,
    JobExecutionHours = 10,
    JobExecutionMinutes = 8
};
scheduler.ScheduleJob(deleteExpiredEventsJob, deleteExpiredEventsExecutionSettings);

var crawlEventsExecutionSettings = new JobExecutionSettings()
{
    DbConnectionString = dbConnectionString,
    BaseAddress = baseAddress,
    FacebookToken = facebookToken,
    AppId = "540747676302325",
    AppSecret = "fdd0d6de400fcf26f5a1c79587876c1f",
    JobPeriodType = JobPeriodType.Day,
    JobPeriod = 1,
    JobExecutionHours = 10,
    JobExecutionMinutes = 9
};
scheduler.ScheduleJob(crawlEventsJob, crawlEventsExecutionSettings);
```

В зазначеному фрагменті коду створюються налаштування для виконання двох задач CrawlEventsJob та DeleteExpiredEventsJob на планувальнику. Далі ці налаштування та екземпляри класів задач передаються в якості параметрів у метод ScheduleJob класу JobScheduler, що надає можливість виконувати задачі на планувальнику з періодичністю вказаною в налаштуваннях задач.

4.7.4 Структура системних таблиць бази даних планувальника

Структура системних таблиць, що містять дані про поточне виконання задач на планувальнику містить наступні основні таблиці:

BLOB_TRIGGERS – трігери завдань планувальника, що виконують обробку файлів та відносяться до blob трігерів.

CALENDARS – календарі планувальника, які використовуються для виконання задач за розкладом.

CATCHED_QUERIES – запити до БД, що виконуються в поточний момент на планувальнику під час виконання завдань.

CRON_TRIGGERS – трігери планувальника що виконують задачі за наданим cron виразом.

4.7.4 Структура системних таблиць бази даних планувальника

Структура системних таблиць, що містять дані про поточне виконання задач на планувальнику містить наступні основні таблиці:

BLOB_TRIGGERS – трігери завдань планувальника, що виконують обробку файлів та відносяться до blob трігерів;CALENDARS – календарі планувальника, які використовуються для виконання задач за розкладом.

CATCHED_QUERIES – запити до БД, що виконуються в поточний момент на планувальнику під час виконання завдань.

CRON_TRIGGERS – трігери планувальника що виконують задачі за наданим cron виразом.

FIRED_DUPLICATES – дублі задач, що в даний момент виконуються на планувальнику. Дана таблиця використовується для визначення задач, що паралельно виконуються планувальником.

FIRED_TRIGGERS – таблиця, що містить трігери, які виконуються планувальником у заданий момент часу.

JOB_DETAILS – деталі виконання задач планувальником. Фактично таблиця містить всі необхідні налаштування, що потрібні для виконання задач планувальником.

LOCKS – блокування, що виникають при паралельному виконанні задач на різних app-серверах. Фактично ця таблиця використовується для діагности наявних блокувань при умові неправильного проектування завдань планувальника, що блокують одне одного.

LOGS – таблиця, що містить логи планувальника.

TRIGGERS таблиця, що містить трігери планувальника.

ВИСНОВКИ ДО РОЗДІЛУ

У цьому розділі було проведено формування основних вимог та функціоналу для бібліотеки. Розроблені можливі сценарії використання та визначені, можливі, обмеження для них, що повинно покращити взаємодію систем-клієнтів бібліотеки, а також запобігти виникненню нестандартних ситуацій.

За результатами розглянутих вище досліджень було побудовано діаграми варіантів використання (англ. use-case) для основних прецедентів.

В рамках розділу виконано порівняння монолітної та мікро-сервісної архітектур. Визначено переваги та недоліки кожного підходу та прийнято рішення використання мікро-сервісного підходу при розробці бібліотеки планувальника.

З огляду на обрану архітектуру розробки проаналізовано та обрано технології розробки бібліотеки та спроектовано архітектуру мікро-сервісів бібліотеки, визначено платформу їх розгортування та перелік допоміжних бібліотек. В якості технологій розробки обрано платформу .NET Core та PostgreSQL у якості база даних для зберігання інформації про поточне виконання задач, їх розклади та статуси виконання. В якості платформи розгортування, що буде підтримуватись бібліотекою «за замовчуванням» було обрано контейнери Docker та Kubernetes у якості модуля оркестрації контейнерів бібліотеки. Було розроблено та описано структуру основних класів кожного з мікро-сервісів, а також структуру таблиць БД, що зберігають конфігураційні дані про задачі, що виконуються планувальником, а також про runtime дані виконання задач.

5 РОЗРОБЛЕННЯ СТАРТАП ПРОЕКТУ

5.1 Опис ідеї проекту

Планувальник задач для розподілених Enterprise систем є продуктом, що має широкий спектр клієнтів, від невеликих систем автоматизації до Enterprise систем до яких адаптовано всі потужності планувальника. Основні напрямки застосування та плюси від використання інноваційних рішень, запропонованих планувальником можна розглянути в табл. 5.1.

Таблиця 5.1 Зміст ідей стартап проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Подання конфігурації планувальника у зрозумілому для користувача вигляді	Системи SOHO та SMALL сегментів, які можуть адмініструвати аналітики	Спеціаліст, що не має технічної освіти, та виконує роль користувача системи, в якій застосовується планувальник, може легко конфігурувати його за допомогою зрозумілого йому подання інформації та документації
Застосування генетичного алгоритму створення розкладу виконання задач	Математичні моделі та алгоритми – алгоритми планування. Вирішення задач оптимізації	Користувач, використовуючи планувальник у кінцевій системі, отримує максимально рівномірне навантаження на вузли системи, не турбуючись про те, що hardware частина буде простоювати
Надання можливості роботи у режимі web-ферми	Системи Enterprise сегментів	Системи, що працюють з великими обсягами даних, і мають розподілену архітектуру, зможуть ефективно застосовувати свої потужності і не втрачати дані, що будуть оброблюватись планувальником на різних app серверах

Таблиця 5.1 Зміст ідей стартап проекту

Надання можливості конфігурування кожного екземпляра планувальника в розподіленій системі окремо	Адміністрування продуктів в системах Enterprise рівня	Кінцевий користувач зможе гнучко конфігурувати планувальник з огляду на потужності та застосування кожного з вузлів розподіленої системи

З техніко-економічної точки зору продукт має ряд переваг. Проаналізуємо основні техніко-економічні характеристики продукту. З точки зору бар'єрів проникнення на ринок, основними факторами продукту можна вважати те, що він є open-source продуктом, а отже не потребує вкладень з боку покупця на встановлення ліцензій. Також використовуючи продукт покупець зможе зменшити витрати на виробництво, з огляду на технічні властивості продукту, щодо оптимізації обчислювальних ресурсів. Галузь, в якій просувається продукт є дуже динамічної і з огляду на це слід зазначити ще декілька техніко-економічних характеристик, що дозволяють мати перевагу над конкурентами, а саме – інформаційне забезпечення, та розвиток лінійки продукту.

Отже, розглянувши загальний перелік техніко-економічних властивостей продукту перейдемо до аналізу сильних, слабких та нейтральних характеристик ідеї в порівнянні з конкурентами, що зазначені в табл. 5.2.

Таблиця 5.2 Аналіз характеристик ідеї в порівнянні з конкурентами

Техніко-економічні характеристики ідеї	Мій проект	Hang-fire	Quartz	Fluent Sched-uler	W	N	S
Початкові вкладення	Відсутні	Присутні	Присутні	Відсутні			Порівняно з найбільш потужними конкурентами початкові вкладення відсутні
Техніко-економічні характеристики ідеї	Мій проект	Hang-fire	Quartz	Fluent Sched-uler	W	N	S

Таблиця 5.2 Аналіз характеристик ідеї в порівнянні з конкурентами

Витрати на виробництво	Зменшені	Зменшені	Зменшені	Незмінні			Витрати на виробництво зменшені, але в конкурентів маємо такі ж показники
Інформаційне забезпечення	Повне	Часткове	Часткове	Часткове			Порівняно з конкурентами продукт має повне інформаційне забезпечення
Розвиток продуктової лінійки	Присутній	Присутній	Присутній	Відсутній		Продукти-конкуренти розвивають свою лінію продуктів, отже характеристика - нейтральна	

З огляду на конкурентні продукти, запропонований продукт має ряд переваг у важливих техніко-економічних характеристиках і не має негативних характеристик.

5.2 Технологічний аудит ідеї проекту

Проведемо аудит технології, за допомогою якої можна реалізувати ідею продукту. Для цього визначимо технології виготовлення продукту, існування таких технологій, або стадію їх розробки та доступність технології. Ці показники зазначено в табл 5.3.

Таблиця 5.3 Технологічна здійсненність ідеї проекту

Ідея проекту	Технології реалізації	Наявність технології	Доступність технології
Подання конфігурації планувальника у зрозумілому для користувача вигляді	XML, JSON серіалізація за допомогою бібліотек серіалізації	Наявна технологія	Open-source бібліотеки
Застосування генетичного алгоритму створення розкладу виконання задач	Інструменти розобки на платформі .NET Core	Інструменти та технології розробки наявні	Доступні Express версії продуктів

Таблиця 5.3 Технологічна здійсненність ідеї проекту

Надання можливості роботи у режимі web-ферми	Використання бібліотек та продуктів балансування навантаження	Технології наявні	Наявні trial версії для розробки
Надання можливості конфігурування кожного екземпляра планувальника в розподіленій системі окремо	XML, JSON серіалізація за допомогою бібліотек серіалізації. Засоби поєднання конфігурацій	Наявна технологія	Open-source бібліотеки

Всі необхідні технології та інструменти для розробки продукту є наявними на ринку та доступними на час розробки продукту. Таким чином технологічна реалізація продукту можлива і не має жодних перешкод.

5.3 Аналіз ринкових можливостей запуску стартап-проекту

Визначимо ринкові можливості, які можна використати під час ринкового впровадження проекту, та ринкові загрози, які можуть перешкодити реалізації проекту. Зазначений аналіз дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів. Результати аналізу попиту на продукт, що розробляється, зазначено в табл. 5.4.

Таблиця 5.4 Аналіз попиту на продукт

Показники стану ринку	Характеристика
Кількість головних гравців, од	5-7
Загальний обсяг продаж грн/ум.од	10-15 тис. / місяц
Динаміка ринку	Зростає
Наявність обмежень для входу	Наявність конкурентів
Специфічні умови для стандартизації та сертифікації	Відсутні
Середня норма рентабельності в галузі	65-70%

Середня норма рентабельності значно виза за банківські відсотки вкладення, отже проект є привабливим для інвестицій за попереднім оцінюванням. Визначимо потенційні групи клієнтів, їх характеристики та сформуємо орієнтовний перелік вимог до товари для кожної групи (табл. 5.5).

Таблиця 5.5 Потенційні групи клієнтів

Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці різних цільових груп	Вимоги споживачів до товару
Потреба виконання бізнес-задач за розкладом	SOHO,SMALL,MEDIUM,LARGE,ENTERPRISE компанії, що використовують різноманітні системи автоматизації своїх процесів	Кожна з груп має свої сценарії використання продукту та свої вимоги до швидкості і якості надання результатів виконання	Зручна конфігурація; Можливість використання у веб-фермі; Зручне розгортування;

Виконаємо аналіз ринкового середовища: визначемо групи факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають (табл. 5.6, табл. 5.7).

Таблиця 5.6 Фактори загроз

Фактор	Зміст загрози	Можлива реакція компанії
Зміни у податках	Зміни сплати податків, що значно впливають на фінансову модель	Застосування іншої фінансової моделі, технології розповсюдження продукту
Помилки апаратури	Непередбачені помилки в роботі апаратури, що впливають на строки виконання	Закупівля якісного обладнання для розробки
Піратство	Злами ПЗ, що впливають на розповсюдження продукту та фінансову модель	Покращення системи ліцензування продукту
Зміни законодавства	Зміни законодавства, що значно впливають на фінансову модель	Застосування іншої фінансової моделі, технології розповсюдження продукту
Несприйняття ідеї користувачами	Несприйняття ідеї продукту, низька зацікавленість та попит з боку користувачів	Зміна підходу до реклами, менеджменту корисутвачів

Таблиця 5.7 Фактори можливостей

Фактор	Зміст можливості	Можлива реакція компанії
Розвиток ринку open-source ПЗ	Розвиток ринку open-source ПЗ та поява нових бібліотек, які можна застосувати для покращення розробок	Застосування open-source бібліотек у продуктів
Залучення outsource компаній до розробки	Залучення outsource компаній(спеціалістів) у процес розробки ПЗ	Залучення спеціалістів у процес розробки. Насамперед архітекторів та досвідчених спеціалістів для покращення якості продукту

Таблиця 5.7 Фактори можливостей

Розвиток hardware	Розвиток hardware, що впливає на потужності на яких працює продукт	Пришвидшення розробки, покращення якості виконання та швидкості тестування

На основі зазначених факторів ризику та можливостей можна провести аналіз пропозиції та визначити загальні риси конкуренції на ринку.

Таблиця 5.8 Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентноспроможною)
Олігополія чиста	У кількості постачальників ПЗ до користувачів	Велика конкуренція, розробка рішення, що має новизну
Національний рівень конкурентної боротьби	Розробка ПЗ не прив’язана до конкретного регіону	Особливість середовища не впливає на діяльність підприємства
Внутрішньогалузева конкуренція	Конкуренція між продуктами спостерігається всередені галузі	Необхідно розробити продукт, що має високі конкуретноспроможні показники в галузі
Товарно-видова конкуренція	Конкуренція спостерігається між різними видами товарів(продуктів)	Необхідно розробити продукт, що якісно відрізнається від інших видів, або ж схожих видів
Цінова конкурентна перевага	Переваги в галузі досягаються не лише за рахунок кращого результату використання продуктів, але і за рахунок цінових переваг	Продукт розробляється як open-source що надає максимальну цінову перевагу при виборі рішення клієнтами
Інтенсивність не марочна	Не спостерігається марочної інтенсивності	Не впливає на діяльність підприємства

За результатами аналізу слід зазначити, що робота на ринку з огляду на конкурентну ситуацію можлива і практично не має перешкод. Необхідними умовами входження на ринок є позиціонування продукту як open-source

рішення. Надання якісних переваг з боку адміністрування продукту, його підтримки та наявності інформаційного забезпечення. Зазначені вище фактори дозволяють зробити продукт конкурентоспроможним на ринку з огляду на переваги та недоліки порівняно з конкурентами (табл. 5.9).

Таблиця 5.9 Обґрунтування факторів конкурентоспроможності

Фактор	Обґрунтування
Ціна	Продукт позиціонується як open-source, що надає переваги перед конкурентами з точки зору цінового фактору вибору
Зручність адміністрування	Технічна перевага продукту так як конкуренти мають складні механізми розгортування та складно адмініструються
Зручність налаштування	Технічна перевага продукту так як конкуренти мають складні механізми налаштування та потребують залучення спеціалістів з налаштування ПЗ зазначеного типу
Робота в режимі web-ферми	Перевага продукту для клієнтів enterprise сегменту, оскільки продукти конкурентів не адаптовані для enterprise ринку взагалі

Зазначені фактори дозволяють продукту бути конкурентоспроможним на ринку, але кожен з факторів має різний рівень впливу на успіх продукту на ринку в цілому, в табл. 5.10 зазначені сильні та слабкі сторони продукту у формі порівняння, що дозволяє визначити найбільш важливі з них. Скорочені назви конкурентів, що містяться в таблиці: Q-Quartz, H-Hangfire, F – FluentScheduler.

Таблиця 5.10 Порівняльний аналіз сильних та слабких сторін «Планувальника для розподілених Enterprise систем»

Фактор	Балл	-3	-2	-1	0	1	2	3
Ціна	18			Q,H	F			
Зручність адміністрування	15	F			Q,H			
Зручність налаштування	15	Q	H		F			
Інформаційне забезпечення	10	Q,F		H				
Робота в режимі web-ферми	20	F		H,Q				
Можливість кастомізації	5	F				Q,H		

Таблиця 5.11 SWOT-аналіз стартап проекту

Сильні сторони: Ціна, Зручність адміністрування, Зручність налаштування, Інформаційне забезпечення, Робота в режимі web-ферми	**Слабкі сторони:** Можливість кастомізації
Можливості: Створення бібліотек інструментів адміністрування розподілених Enterprise рішень в якості окремого продукту. Залучення користувачів до створення бази знань, що значно покращить інформаційне забезпечення. Створення окремих рішень з налаштування системи, що зробить її більш гнучкою та дозволить виконувати оновлення ПЗ і додавання нових можливостей і налаштувань у runtime.	**Загрози:** Поява широкої потреби в кастомізації рішення, яка буде супроводжуватись наявністю великого переліку нових задач до розробки, які не будуть відноситись до цільової аудиторії загалом, а лише до конкретних клієнтів. Це приводить до того що немає чіткого бачення стратегії розвитку продукту, а отже перешкоджає якісному плануванню розвитку проекту в майбутньому.

Таблиця 5.11 містить SWOT аналіз продукту, що розробляється та надає змогу розглянути сильні і слабкі сторони продукту, а також можливості та загрози, що існують в процесі розробки стартап проекту. Альтернативою ринкового впровадження стартап проекту з огляду на наявну загрозу у зростанні необхідності кастомізації продукту для конкретних клієнтів є: залучення окремих проектних команд для кожного клієнта, що потребує більш гнучкої розробки продукту для своїх потреб. Даний підхід дозволить мати чітку стратегію розвитку продукту в цілому і задовольнить потреби окремих клієнтів, що бажають виконати особливі кастомізації продукту для своїх потреб бізнесу.

5.4 Розроблення ринкової стратегії проекту

Основою ринкової стратегії проекту є опис цільових груп потенційних користувачів продукту, який зазначено в табл. 5.12.

Таблиця 5.12 Цільові групи користувачів

Опис профілю цільової групи	Готовність сприйняти продукт	Орієнтовний попит	Інтенсивність конкуренції	Простота входу
SOHO компанії	Низька	Низький	Висока	Висока
SMALL компанії	Низька	Низький	Висока	Висока

Таблиця 5.12 Цільові групи користувачів

MEDIUM компанії	Середня	Низький	Висока	Висока
LARGE компанії	Середня	Середній	Висока	Середня
ENTERPRISE компанії	Висока	Високий	Середня	Середня

За результатами аналізу цільових груп потенційних користувачів слід зазначити, що найбільш вигідним сегментом є Enterprise компанії оскільки користувачі цього сегменту мають високий попит і готовність сприйняти продукт. Отже, оскільки компанія зосереджується на одному сегменті то слід обрати стратегію концентрованого маркетингу. Базова стратегія конкурентної поведінки включає такі положення: проект не є «першопрохідцем» на ринку, а отже надає рішення, що має переваги над конкурентами але не є цілком унікальним; проект буде шукати нових споживачів, розширюючи клієнтський портфель, але буде проводити заходи з залучення нових клієнтів шляхом переходу від конкурентів; оскільки продукт не є цілком новим, то основні характеристики планувальників, за винятком унікальних характеристик пов’язаних з перевагами рішення будуть однакові з конкурентами; в якості стратегії буде обрана стратегія зайняття конкурентної ніші;

5.5 Розроблення маркетингової програми стартап-проекту

Розглянемо основні вигоди, що пропонує стартап проект для вирішення потреб користувачів та переваги рішення перед конкурентами (табл. 5.13)

Таблиця 5.13 Ключові переваги проекту

Потреба	Вигода	Переваги над конкурентами
Подання конфігурації планувальника у зрозумілому для користувача вигляді	Системи SOHO та SMALL сегментів, які можуть адмініструвати аналітики	Спеціаліст, що не має технічної освіти, та виконує роль користувача системи, в якій застосовується планувальник, може легко конфігурувати його за допомогою зрозумілого йому подання інформації та документації

Таблиця 5.13 Ключові переваги проекту

Застосування генетичного алгоритму створення розкладу виконання задач	Математичні моделі та алгоритми – алгоритми планування. Вирішення задач оптимізації.	Користувач, використовуючи планувальник у кінцевій системі, отримує максимально рівномірне навантаження на вузли системи, не турбуючись про те, що hardware частина буде простоювати
Надання можливості роботи у режимі web-ферми	Системи Enterprise сегментів.	Системи, що працюють з великими обсягами даних, і мають розподілену архітектуру, зможуть ефективно застосовувати свої потужності і не втрачати дані, що будуть оброблюватись планувальником на різних app серверах.

Збут продукту виконується власними силами за допомогою активної участі у Open-source community, шляхом створення landing сторінок продукту, контекстної реклами.

ВИСНОВКИ ДО РОЗДІЛУ

В результаті проведення аналізу можливості створення комерціалізації та подальшого розвитку стартап проекту, проведено технологічний аудит стартап проекту, розроблено основні ринкові стратегії запуску проекту, стратегії конкурентоспроможності проекту, SWOT аналіз. Визначено маркетингову стратегію проекту, сильні та слабкі фактори порівняно з конкурентами а також чітко визначено рівень впливу кожного з факторів на розвиток проекту в цілому.

Слід зазначити, що проект має великий попит серед конкурентної ніші користувачів обраного сегменту, рентабельність роботи на ринку в рамках обраного сегменти є високою, а отже проект є привабливим для інвестицій, саме тому прийнято рішення використовувати стратегію концентрованого маркетингу в поєднанні з стратегією зайняття конкурентної ніші.

З огляду на потенційні групи клієнтів та існуючі бар’єри для входження а також стан конкуренції слід зазначити, що проект має перспективи впровадження та позитивні прогнози з боку конкурентоспроможності продукту на ринку в цілому. Проект має певні складності з боку негативного фактору необхідності кастомізації продукту для конкретних клієнтів, що негативно впливає на стратегію розвитку продукту. Альтернативним рішенням цієї проблеми визначено застосування проектних команд для впровадження рішення у клієнта. Таким чином подальша імплементація проекту є повністю доцільною.

6 МОДЕЛЮВАННЯ АЛГОРИТМУ ПЛАНУВАННЯ

6.1 Моделювання та генерація набору завдань

Для отримання результатів роботи алгоритму проводилося 15-20 запусків алгоритму на кожному наборі задач. Було проведено ряд експерименті для виявлення найбільш оптимальних параметрів роботи генетичного алгоритму.

В якості генератора набору задач використовувалася програмний інструментарій генерації вхідних даних для генетичних алгоритмів generate. Використаний інструментарій є рекомендованою бібліотекою для генерації наборів задач та тестування генетичних алгоритмів. Для формування набору даних використовуються наступні параметри:

a - стиль появлення задач у потоці;

s - кількість елементарних машин у системі, для якої генерується набір;

j - кількість задач у наборі;

r - відсоток масштабованих задач у наборі;

o- ім'я файлу, в який потрібно записати набір.

Для дослідження роботи алгоритму згенеровано набори завдань з різними об’ємами, різним відсотком масштабованості і для різного розміру обчислювальної системи.

6.2 Результати моделювання

Для дослідження залежності часу рішення набору задач від проценту масштабованості були створені 4 набори для 12 елементарних машин з 500 задач з 0, 25, 50, 75 і 100 відсотками масштабованості.

Гістограма, яка порівнює результати роботи генетичного алгоритму та алгоритму NFDH зображена на рис. 6.1. Можна зробити висновок, що властивість масштабованості істотно впливає на значення цільової функції, як генетичного алгоритму, так і NFDH.

Найкращій результат пошуку оптимального рішення досягається тоді, коли кожна задача в наборі масштабована. У випадку, коли немає масштабованих задач, генетичний алгоритм не має ніяких переваг і, в цілому не є оптимальним. При 25 відсотках масштабованих завдань в наборі поліпшення незначне. При 50% можна спостерігати істотне поліпшення.

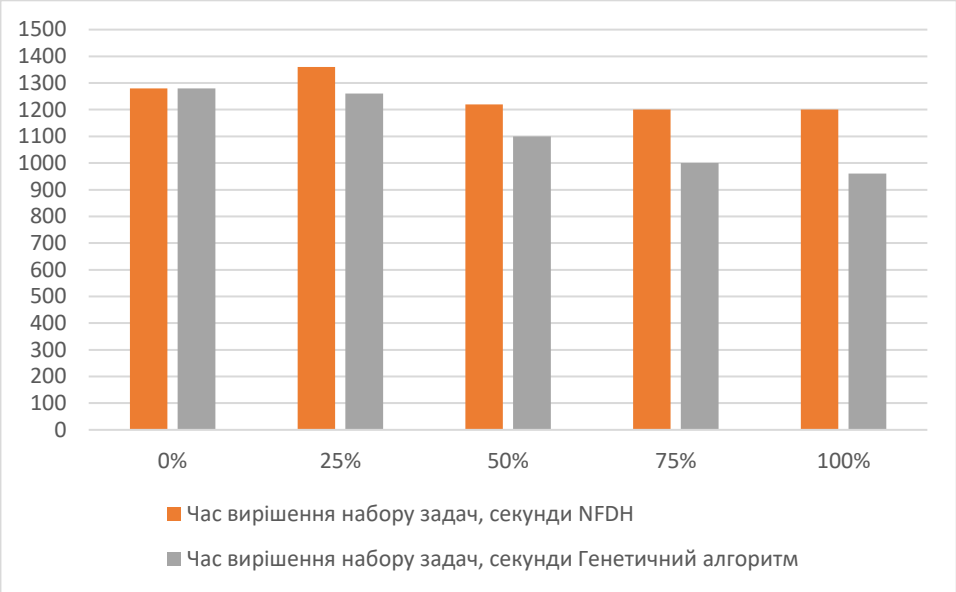


Рисунок 6.1 Порівняння результатів роботи генетичного алгоритму та алгоритму NFDH

На рис. 6.2 приведена гістограма залежності часу рішення набору задач від розміру популяції. Для цього експерименту згенеровано набори з 800 задач, для 9 елементарних машин, відсоток масштабованих задач - 100. Виходячи з отриманих даних, слід сказати, що найбільш висока якість розкладу створюється при максимально великому розмірі популяції. Але, тут є істотний недолік, а саме - час роботи алгоритму, оскільки обчислювальна складність безпосередньо залежить від кількості осіб популяції. При розмірі популяції, рівної 4500 особам, алгоритм досягає найкращого результату, але час роботи алгоритм не виправдовує цього поліпшення, тому 1500 осіб є рекомендованим розміром популяції.

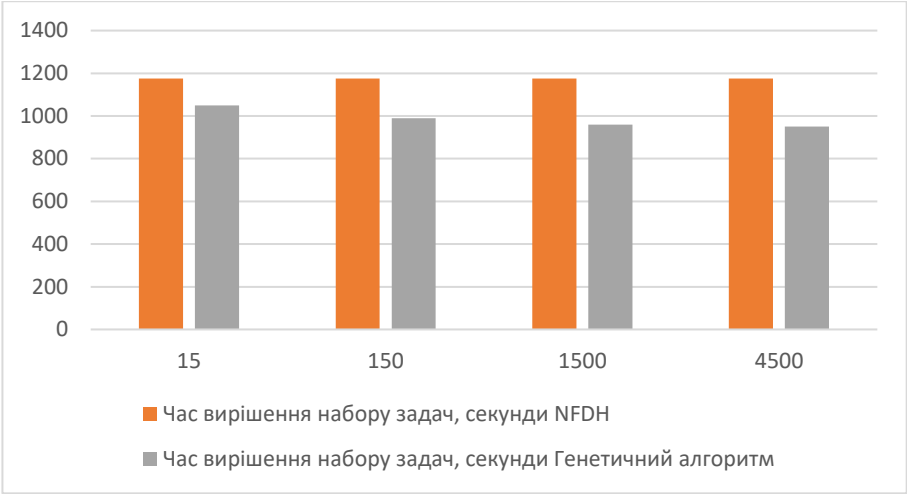


Рисунок 6.2 Залежність часу рішення набору задач від розміру популяції

Для наступного експерименту було згенеровано по 4 набори для обчислювальних систем розміром 9, 18 і 36 обчислювальних машин з 50, 200, 400 і 800 задач. За результатами роботи алгоритму з даними наборами побудований графік, зображений на рис. 6.3. На невеликих наборах задач використання великих обчислювальних систем не має переваг перед невеликими обчислювальними системами. Для обробки великого набору задач розумніше використовувати більші масштабні обчислювальні системи.

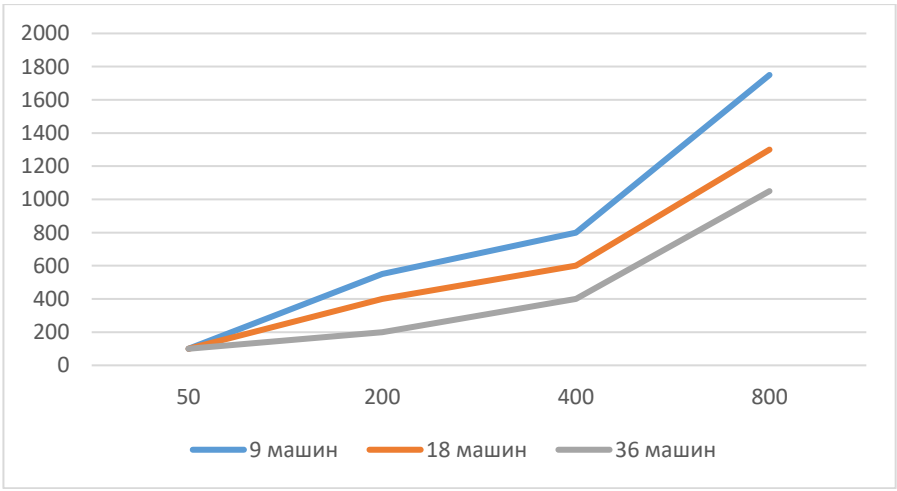


Рисунок 6.3 Залежність часу роботи генетичного алгоритму від масштабу системи

Як вказано вище, деякі параметри не доцільно використовувати при роботі з генетичним алгоритмом, так як з ними програма буду будувати розклад задач за досить великий час. Тестувались набори з 100, 200, 400, 800 і 1600 задач і вимірюється час роботи NFDH і генетичного алгоритму.

Час роботи NFDH істотно менше часу виконання генетичного алгоритму, але якість результуючого розкладу значно вище. Великі набори генетичний алгоритм обробляє істотно довше. Якщо користувачу не критично таке очікування, то генетичний алгоритм рекомендується використовувати на великих наборах задач.

Виходячи з проведених досліджень, можна скласти список рекомендованих параметрів, з якими генетичний алгоритм планування буде демонструвати найкращі результати. Рекомендоване число поколінь - 15. В більшості випадків після 15-го покоління не спостерігається зміна цільової функції. При розмірі популяції, більше, ніж 1500 одиниць, досягається кращій результат, але час роботи алгоритму не виправдовує цього поліпшення, тому рекомендований розмір популяції - 1500 осіб. Чим більше відсоток масштабованих завдань в наборі, тим краще значення цільової функції.

6.3 Випробування бібліотеки планувальника

Випробування бібліотеки планувальника проводилося у реальній, навантаженій Enterprise CRM системі, що має наступну архітектуру:

6 app-серверів з балансувальником навантаження Haproxy в режимі round-robin;

replica-set бази даних, БД мають як дані спільні для всіх app-серверів, так і унікальні для певного app-сервера;

2 сервери кешування Redis.

Для демонстрації роботи бібліотеки планувальника розглянемо налаштування екземплярів планувальника на кожному з app-серверів, виконаємо запуск планувальника і відслідкуємо дані виконання задач на планувальнику. Планувальник налаштований з використанням бази даних для розміщення runtime даних, та xml файлів для конфігурації.

Конфігурація планувальника на кожному з app серверів має схожий вигляд та зображена на рис. 6.4.

```
<scheduler isActive="true">
  <add key="scheduler.scheduler.instanceId" value="AUTO" />
  <add key="scheduler.scheduler.instanceName" value="CampaignschedulerScheduler" />
  <add key="scheduler.threadPool.threadCount" value="5" />
  <add key="scheduler.jobStore.misfireThreshold" value="300000" />
  <add key="scheduler.jobStore.clustered" value="true" />
  <add key="scheduler.jobStore.type" value="scheduler.Impl.AdoJobStore.JobStoreTX, scheduler" />
  <add key="scheduler.jobStore.dataSource" value="SchedulerDb" />
  <add key="scheduler.jobStore.driverDelegateType" value="scheduler.Impl.AdoJobStore.SqlServerDelegate, scheduler" />
  <add key="scheduler.dataSource.SchedulerDb.provider" value="SqlServer-20" />
  <add key="scheduler.dataSource.SchedulerDb.connectionStringName" value="schedulerDb" />
  <add key="scheduler.jobStore.acquireTriggersWithinLock" value="true" />
</scheduler>
<scheduler isActive="false">
  <add key="scheduler.scheduler.instanceId" value="AUTO" />
  <add key="scheduler.scheduler.instanceName" value="PortalschedulerScheduler" />
  <add key="scheduler.threadPool.threadCount" value="2" />
  <add key="scheduler.jobStore.misfireThreshold" value="300000" />
  <add key="scheduler.jobStore.clustered" value="true" />
  <add key="scheduler.jobStore.type" value="scheduler.Impl.AdoJobStore.JobStoreTX, scheduler" />
  <add key="scheduler.jobStore.dataSource" value="SchedulerDb" />
  <add key="scheduler.jobStore.driverDelegateType" value="scheduler.Impl.AdoJobStore.SqlServerDelegate, scheduler" />
  <add key="scheduler.dataSource.SchedulerDb.provider" value="SqlServer-20" />
  <add key="scheduler.dataSource.SchedulerDb.connectionStringName" value="schedulerDb" />
  <add key="scheduler.jobStore.acquireTriggersWithinLock" value="true" />
</scheduler>
</schedulerConfig>
```

Рисунок 6.4 Конфігурація планувальників на app серверах

Бачимо, що на app серверах створено по декілька екземплярів планувальника, але на 4 з 6 application серверах планувальник CampaignschedulerScheduler є активним, на інших app –серверах планувальник є неактивним. PortalschedulerScheduler є неактивним на 4 app серверах та є активним на портальних app серверах.

Таким чином бачимо потенціальну проблему блокування завдань від планувальників, що є активними на декількох app серверах у розподіленій системі. Розглянемо runtime дані виконання задач на планувальнику за допомогою запиту до системної таблиці планувальника. Текст запиту для перегляду активних задач наведено нижче:

```
SELECT
       SCHED_NAME,
       CAST((FIRED_TIME) / 864000000000.0 - 693595.0 AS DATETIME) +
(GETDATE() - GETUTCDATE()) FIRED_TIME,
       CAST((SCHED_TIME) / 864000000000.0 - 693595.0 AS DATETIME) +
(GETDATE() - GETUTCDATE()) SCHED_TIME,
        ENTRY_ID,
        TRIGGER_NAME,
        TRIGGER_GROUP,
        INSTANCE_NAME,
        PRIORITY,
        STATE,
        JOB_NAME,
```

```
        JOB_GROUP
    FROM QRTZ_FIRED_TRIGGERS
```

Розглянемо перелік даних, що надають інформацію про поточну ситуацію у виконанні задач на планувальнику завдань:

1. SCHED_NAME – назва планувальника на якому було запущено завдання.
2. FIRED_TIME – час запуску завдання, для обробки даного показника використовується формула перетворення даних з формату зберігання Timespan у формат Дата\Час, яку можна бачити у тексті запиту до БД.
3. SCHED_TIME - час встановлення завдання на планувальник, для обробки даного показника використовується формула перетворення даних з формату зберігання Timespan у формат Дата\Час, яку можна бачити у тексті запиту до БД.
4. ENTRY_ID – адреса app-сервера з ключем планувальника, використовується для ідентифікації середовища запуску задачі.
5. TRIGGER_NAME – назва трігеру, що запустив задачу.
6. TRIGGER_GROUP – назва групи трігерів до якої відноситься трігер, який запустив задачу.
7. INSTANCE_NAME – унікальна назва екземпляру app сервера, на якому була запущена задача.
8. PRIORITY – пріорітет завдання.
9. STATE – статус задачі на виконанні.
10. JOB_NAME - ім'я джобу планувальника який запускає задачу.
11. JOB_GROUP – назва групи джобів планувальника, до якої відноситься джоб планувальника який запускає задачу;.

Розглянемо результати виконання запиту, що зображено на рис. 6.5. Ми бачимо дані з таблиці стану задач, що виконуються на планувальнику.

SCHED_NAME	FIRED_TIME	SCHED_TIME	ENTRY_ID	TRIGGER_NAME	TRIGGER_GROUP
zScheduler	2018-10-30 23:55:08.927	2018-10-30 23:55:38.720	ts1-portal-web1.tscrm.com636764968864428904636...		BeesenderOperator
zScheduler	2018-10-30 22:42:15.790	2018-10-30 22:42:15.617	tsinternalcrm1.tscrm.com636764605303303142636764...		RECOVERING_JOBS
zScheduler	2018-10-30 22:42:15.883	2018-10-30 22:42:15.650	tsinternalcrm1.tscrm.com636764605303303142636764...		RECOVERING_JOBS
zScheduler	2018-10-30 23:55:04.923	2018-10-30 23:55:14.910	tsinternalcrm1.tscrm.com636764605303303142636764...		Exchange
zScheduler	2018-10-30 23:40:41.333	2018-10-30 23:40:21.647	tsinternalcrm3.tscrm.com636764618990349377636764...		RECOVERING_JOBS
zScheduler	2018-10-30 23:55:04.947	2018-10-30 23:55:27.133	tsinternalcrm3.tscrm.com636764618990349377636764...		Exchange
zScheduler	2018-10-30 23:55:04.847	2018-10-30 23:55:33.123	tsinternalcrm4.tscrm.com636764611925774717636764...		CampaignJobGroup

INSTANCE_NAME	PRIORITY	STATE	JOB_NAME	JOB_GROUP
ts1-portal-web1.tscrm.com636764968864428904	5	ACQUIRED		NULL
tsinternalcrm1.tscrm.com636764605303303142	0	EXECUTING		Exchange
tsinternalcrm1.tscrm.com636764605303303142	0	EXECUTING		OperatorSingleWindow
tsinternalcrm1.tscrm.com636764605303303142	5	ACQUIRED		NULL
tsinternalcrm3.tscrm.com636764618990349377	0	EXECUTING		Exchange
tsinternalcrm3.tscrm.com636764618990349377	5	ACQUIRED		NULL
tsinternalcrm4.tscrm.com636764611925774717	5	ACQUIRED		NULL

Рисунок 6.5 Результати виконання запиту до табиць runtime даних планувальника

Розглянувши результати запиту, бачимо, що INSTANCE_NAME містить назву app-сервера на якому виконується той чи інший джоб планувальника. Ми бачимо, що наприклад джоби групи Exchange виконуються на різних app-серверах та мають статус EXECUTING. Це дає нам змогу бачити, що джоби не блокують один одного та можуть виконуватись на різних app-серверах синхронізуючи результати свого виконання в базі даних.

ВИСНОВКИ ДО РОЗДІЛУ

В даному розділі проведено аналіз швидкодії роботи генетичного алгоритму складання розкладу виконання завдань планувальником. Було виконано порівняння звичайного алгоритму та розробленого генетичного алгоритму за різних умов виконання задачі (кількість задач, відсоток масштабованих задач, кількість серверів у web-фермі). Результатом порівняння швидкодії роботів алгоритму є визначення того, що для великої кількості задач, що необхідно виконати на планувальнику генетичний алгоритм дає значне пришвидшення роботи виконання та синхронізації результатів виконання завдань. Під час тестування алгоритмів, було визначено оптимальні умови виконання генетичного алгоритму, які зазначені в Розділі 5.2. За даних умов генетичний алгоритм дає найбільшу якість та швидкість виконання та складання розкладу задач.

В результаті проведених досліджень роботи бібліотеки планувальника у реальній розподіленій Enterprise системі було виявлено, що задачі з однієї групи можуть виконуватись незалежно на різних app-серверах, синхронізуючи свої результати в БД та не блокуючи одна одну. Таким чином результати впровадження демонструють, що планувальник вирішує поставлене завдання та успішно функціонує в розподіленій системі.

ВИСНОВКИ

У даному дослідженні було проведено огляд та аналіз предметної області планувальників завдань в системах, розглянуто їх основні типи та підходи до реалізації, а також проаналізовано можливості використання планувальників у Enterprise системах з розподіленою архітектурою. На основі підготовчого дослідження предметної області було визначено критичні ділянки існуючих підходів та розглянуто можливі варіанти їх оптимізації.

У рамках дослідження було прийнято рішення про створення власного алгоритму складання розкладу виконання задач планувальником та запропоновано реалізацію вказаного алгоритму. Було прийнято рішення про застосування розробленого алгоритму у власній бібліотеці планувальника до якої було висунуто ряд функціональних вимог:

1) Незалежність від платформи використання.

2) Можливість швидкого налаштування в режимі web-ферми.

3) Висока швидкість обробки задач в режимі web-ферми.

4) Зручний API роботи з планувальником з систем-клєнтів.

Проведено аналіз даних, які можна отримати після створення задач у черзі на виконання планувальником. На основі отриманих даних розроблено генетичний алгоритм створення розкладу виконання задач на планувальнику.

Розроблено генетичний алгоритм побудови оптимального розкладу виконання завдань. Запропонований алгоритм полягає у комбінації алгоритму NDFN та функцій мутації та схрещення. Базова популяція формується за допомогою алгоритму NFDH (Next Fit Decreasing High). Цей алгоритм був обраний через простоту своєї реалізації і досить низької обчислювальної складності. істотним недоліком такого алгоритму є низька якість упаковки. Завдання з набору вибираються випадковим чином, потім сортуються по спадаючій часу рішення. Функція мутації випадковим чином вибирає завдання і змінює її параметри на інші зі списку ресурсних запитів користувача. потім особина перебудовується заново за алгоритмом NFDH. Мутація може як поліпшити якість розкладу, так і погіршити. Імовірність мутації задається в програмі. Після певної кількості життєвих циклів алгоритм зупиняється. Результатом його роботи буде «найкраща» особа з останнього покоління. Таким чином розроблений алгоритм вибору оптимального планування (розкладу)

вирішення задач дозволяє мінімізувати час простою завдання в черзі, та надає змогу пришвидшити процес обробки завдань системою, та більш рівномірно розподіляти завдання між вузлами розподіленої системи, а отже уникнути проблем синхронізації даних та результатів виконання задач.

Також було створено бібліотеку планувальника на платформі .NET Core, що дозволяє досягнути умов cross-платформеності. Розроблена біблотека задовольняє всім поставленим вимогам, має зручний API для підключення систем клієнтів. Бібліотека може бути використана у бідь якій системі-клієнті, що розгортується на будь-якій платформі, ОС, має будь-яку архітектуру ПЗ. Створена бібліотека має ряд переваг над існуючими рішеннями, а саме:

є cross-платформеною;

може розгортуватися і адмініструватися у якості контейнера;

має зручний API інтеграції;

дозволяє розробнику (користувачу) переглядати конфігураційні налаштування планувальника та runtime дані про виконання задач на планувальнику.

Було проведено аналіз швидкодії роботи генетичного алгоритму складання розкладу виконання завдань планувальником. Було виконано порівняння звичайного алгоритму та розробленого генетичного алгоритму за різних умов виконання задачі (кількість задач, відсоток масштабованих задач, кількість серверів у web-фермі). Результатом порівняння швидкодії роботи алгоритму є визначення того, що для великої кількості задач, що необхідно виконати на планувальнику генетичний алгоритм дає значне пришвидшення роботи виконання та синхронізації результатів виконання завдань. Під час тестування алгоритмів, було визначено оптимальні умови виконання генетичного алгоритму, які зазначені в Розділі 5. За даних умов генетичний алгоритм дає найбільшу якість та швидкість виконання та складання розкладу задач. В результаті проведених досліджень роботи бібліотеки планувальника у реальній розподіленій Enterprise системі було виявлено, що задачі з однієї групи можуть виконуватись незалежно на різних app-серверах, синхронізуючи свої результати в БД та не блокуючи одна одну. Таким чином результати впровадження демонструють, що планувальник вирішує поставлене завдання та успішно функціонує в розподіленій системі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Хорошевский, В.Г. Архітектура обчислювальних систем / В.Г. Хорошевський. - М .: Изд-во МГТУ, 2005. - 510 с.

2. Курносов М.Г., Пазников А.А. Основи теорії функціонування розподілених обчислювальних систем. - Новосибірськ: Автограф, 2015.-52

3. Гэри, М. Обчислювальні машини і складні завдання / М. Гері, Д. Джонсон .- М .: Мир, 1982. - 416 с.

4. Лазарев А.А., Гафаров Е.Р., Теорія розкладів. Задачі та алгоритми. - Москва, 2011. - 222 с.

5. Максимова, Е.Н. Паралельний генетичний алгоритм формування розкладання рішень паралельних адаптованих задач на розподілених обчислювальних систем [Текст] / Максимова Є.Н., Мамойленко С.Н., Ефимов А.В. // Інформатика та проблеми телекомунікацій: матеріали Російська науково-технічна конференція. - Новосибірськ: Вид-во ГОУ ВПО «СибГУТИ», 2010, 27 - 28 квітня 2010 року. - Т. 1. - С. 163

6. Шаповалов, Т.С., Планування виконання завдань у розподілених обчислювальних систем з застосуванням генетичних алгоритмів. Хабаровськ, 2010. - 146 с.

7. Євреинов Е.В. Однорідні обчислювальні системи [Текст] / Е.В.Євреїнов, В.Г. Хорошевский - Новосибірськ: Наука, Сибірське відділення, 1978. - 320 с

8. Касавант Т.Л., Куль Дж. Г. Таксономія планування в загальнорозмірній розподіленої обчислювальної системі // IEEE Trans. Про програмне забезпечення. - 1988. - В. 14, № 2.

9. Бруно Дж., Коффман Е.Г., Сеті Р. Планування незалежних завдань щодо скорочення середнього часу закінчення // Зв'язки АСМ. - 1974. - В. 17, № 7.

10. Кофман Е.Г., Грэхем Р.Л. Оптимальне планування для двох процесорних систем // Акта-Електротехніка. - 1972. - В. 1, № 3.

11. Кофман Е.Г., Сеті Р. Алгоритми мінімізації середнього часу потоку // Ата-Електротехніка. - 1976. - В. 6, № 1.

12. Хорн В.А. Мінімізація середнього часу потоку з паралельними машинами // Опера-експлуатаційні дослідження. - 1973. - В. 21, № 3.

13. Клейнрок Л., Никон А.В. Про оптимальні алгоритми планування часових систем // Журнал ACM. - 1981. - В. 28, № 3.

14. Теорія розкладів у обчислювальних машинах / Під ред. Коффмана Э.Ф. - М .: Наука, 1984.

15. Топорков В.В. Рішення конфліктів у пралельних розподілених обчислювальних системах // Автоматика та телемеханіка. - 2003. - № 5.

16. Топорков В.В. Планування та реалізація оптимальних архітектурних рішень у системах обчислення // Автоматика та телемеханіка. - 2001. - № 12.

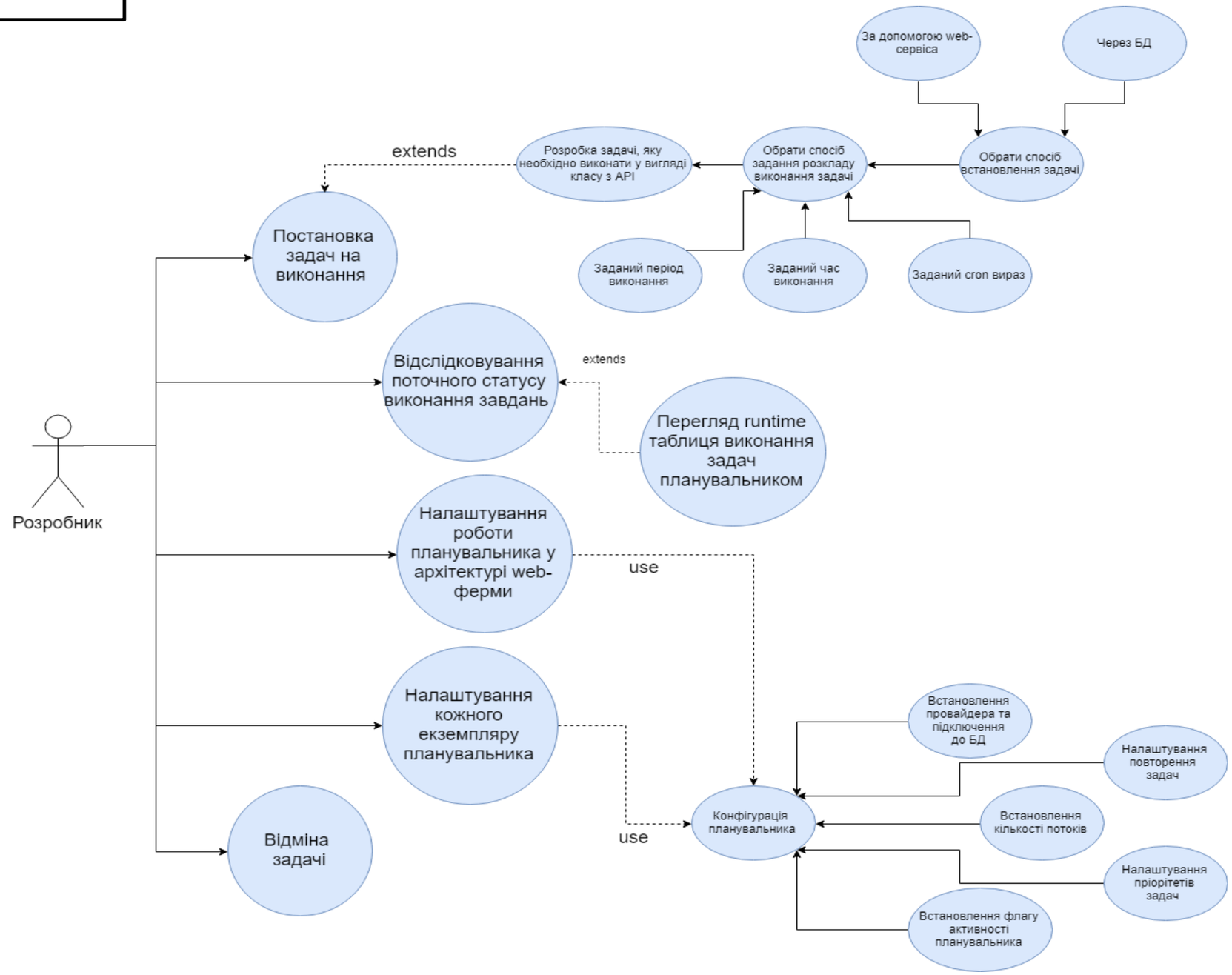
17. Клейнрок Л. Обчислювалні системи з чергами. - М .: Мир, 1979.

18. Михалевич В.С., КуксаА.І. Послідовна оптимізація дискретних мережевих задач оптимального розподілу ресурсів. - М .: Наука, 1983.

19. Хайкин С. Нейронні мережі: повний курс. - М .: Вильямс, 2005.

20.Global Self-Organizing Networks Market Share. Режим доступу: https://medium.com/@kusumrautela21/global-self-organizing-networks-market-share-c93de0efa473

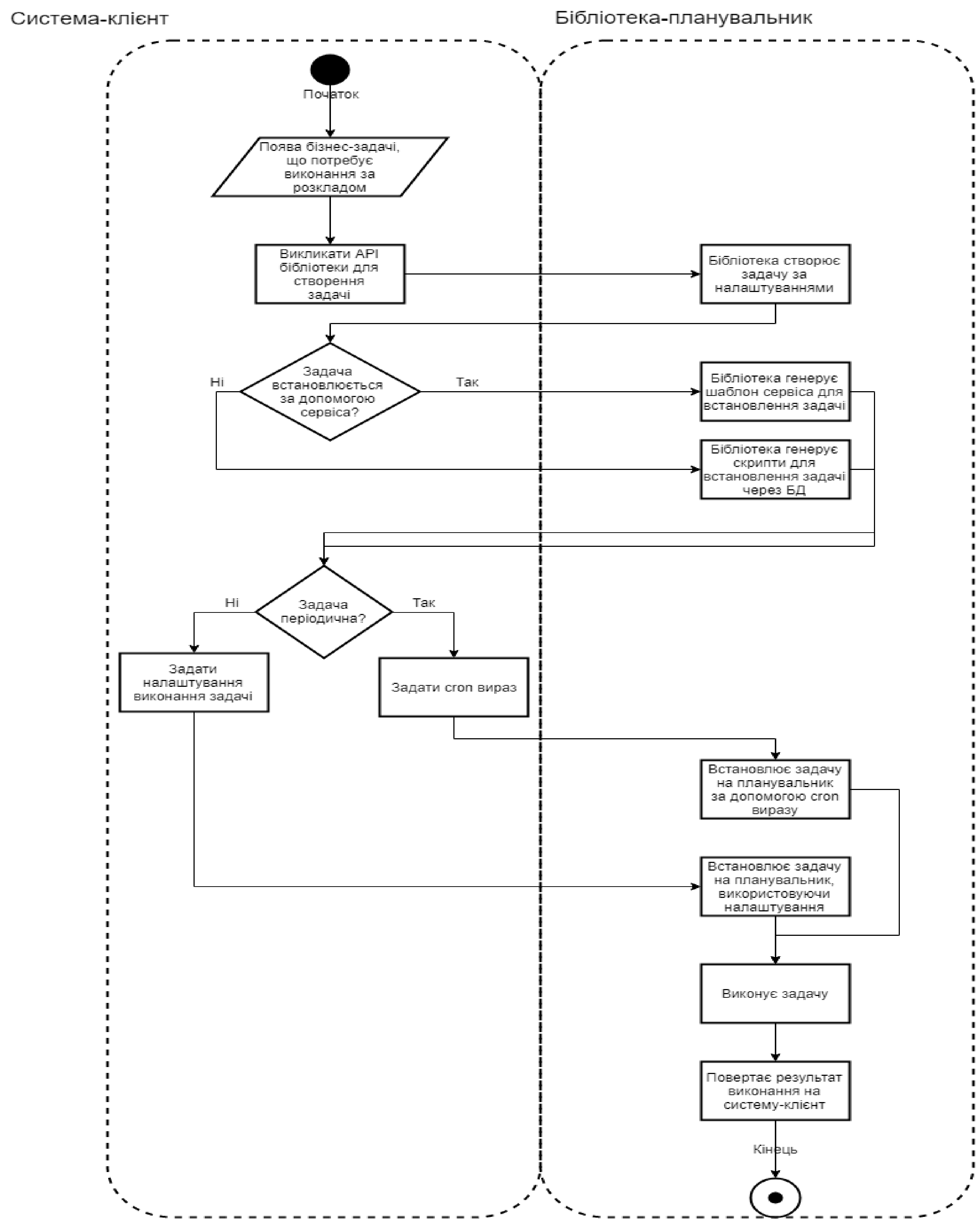
21. Planning and Scheduling Режим доступу: http://www.eetn.gr/index.php/about-eetn/eetn-publications/ai-research-in-greece/planning-and-scheduling



Підпис і дата | Інв. № дубл. | Взам. інв. № | Підпис і дата | Інв. № ориг.

					ІТ-74.191393.02.ССВ	
Зм.	Лист	№ докум	Підпис	Дата	*Демонстраційний плакат «Схема структурна варіантів використання» до магістерської дисертації «Розробка планувальника задач до розподілених Enterprise систем»*	Літ. / Маса / Мірило
Розроб.		Хижняк М.Д.				
Перев.		Пасько В.П.				Лист / Листів
Н. кон.		Пасько В.П.			*Кафедра Технічної кібернетики*	*Група ІТ-74Мп*
Затв.						

ІТ-74.191393.03.ССД



Підпис і дата

Інв. № дубл.

Взам. інв. №

Підпис і дата

Інв. № ориг.

Зм.	Лист	№ докум	Підпис	Дата
Розроб.		Хижняк М.Д.		
Перев.		Пасько В.П.		
Н. кон.		Пасько В.П.		
Затв.				

ІТ-74.191393.03.ССД

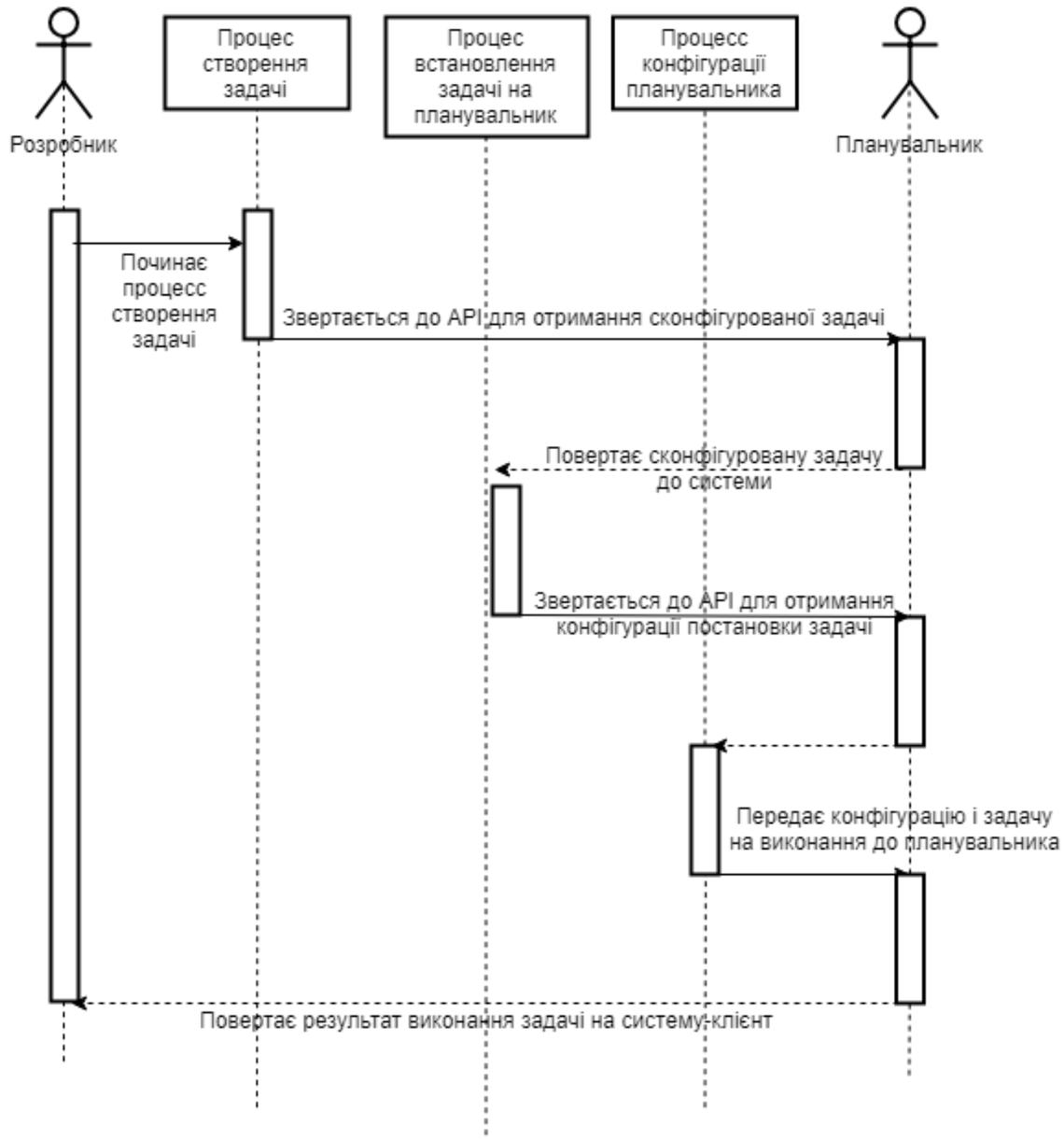
Демонстраційний плакат «Схема структурна діяльності» до магістерської дисертації «Розробка планувальника задач до розподілених Enterprise систем»

Літ.	Маса	Мірило

Лист | Листів

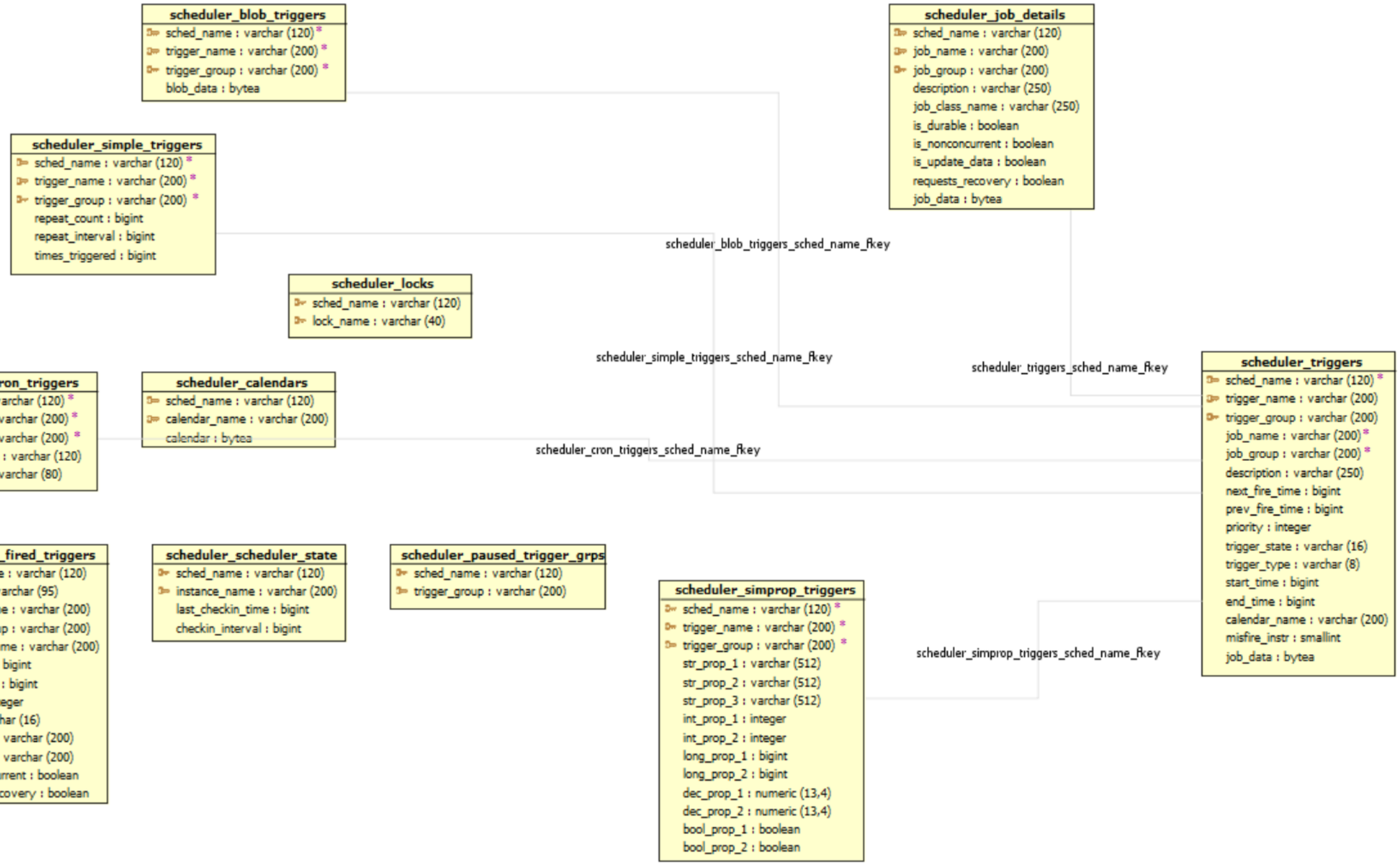
Кафедра Технічної кібернетики

Група ІТ-74мп



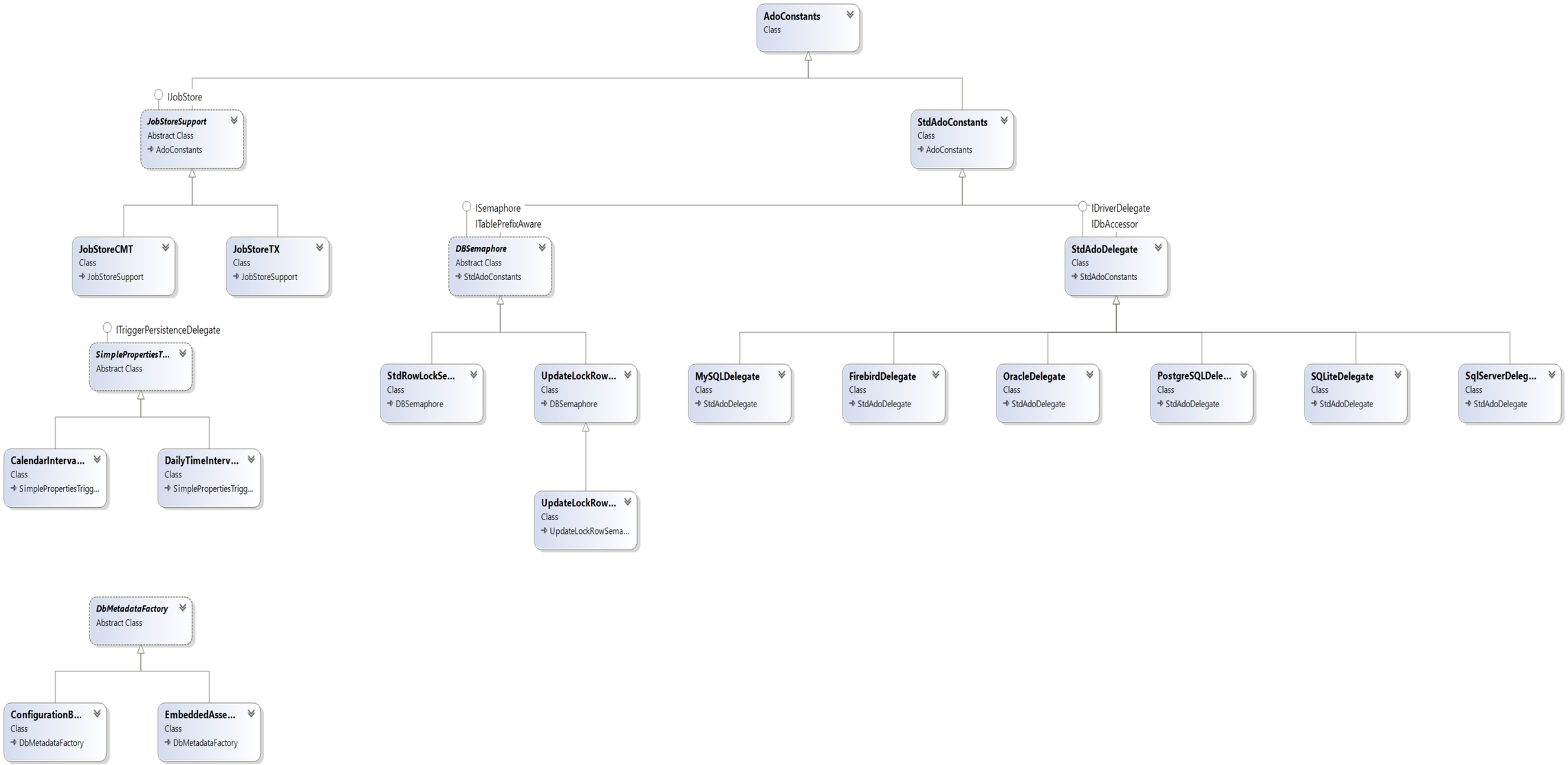
Підпис і дата | Інв. № дубл. | Взам. інв. № | Підпис і дата | Інв. № ориг.

Зм.	Лист	№ докум	Підпис	Дата	ІТ-74.191393.04.ССП		
Розроб.		Хижняк М.Д.			*Демонстраційний плакат «Схема структурна послідовності» до магістерської дисертації «Розробка планувальника задач до розподілених Enterprise систем»*	Літ.	Маса / Мірило
Перев.		Пасько В.П.				Лист	Листів
Н. кон.		Пасько В.П.			*Кафедра Технічної кібернетики*	*Група ІТ-74*	
Затв.							



Підпис і дата | Інв. № дубл. | Взам. інв. № | Підпис і дата | Інв. № ориг.

Зм.	Лист	№ докум	Підпис	Дата	ІТ-74.191393.05.ССБ		
Розроб.		Хижняк М.Д.			*Демонстраційний плакат «Схема даталогічної моделі даних» до магістерської дисертації «Розробка планувальника задач до розподілених Enterprise систем»*	Літ.	Маса / Мірило
Перев.		Пасько В.П.				Лист	Листів
Н. кон.		Пасько В.П.			*Кафедра Технічної кібернетики*	*Група ІТ-74*	
Затв.							



Підпис і дата | Інв. № дубл. | Взам. інв. № | Підпис і дата | Інв. № ориг.

Зм.	Лист	№ докум	Підпис	Дата
Розроб.		Хижняк М.Д.		
Перев.		Пасько В.П.		
Н. кон.		Пасько В.П.		
Затв.				

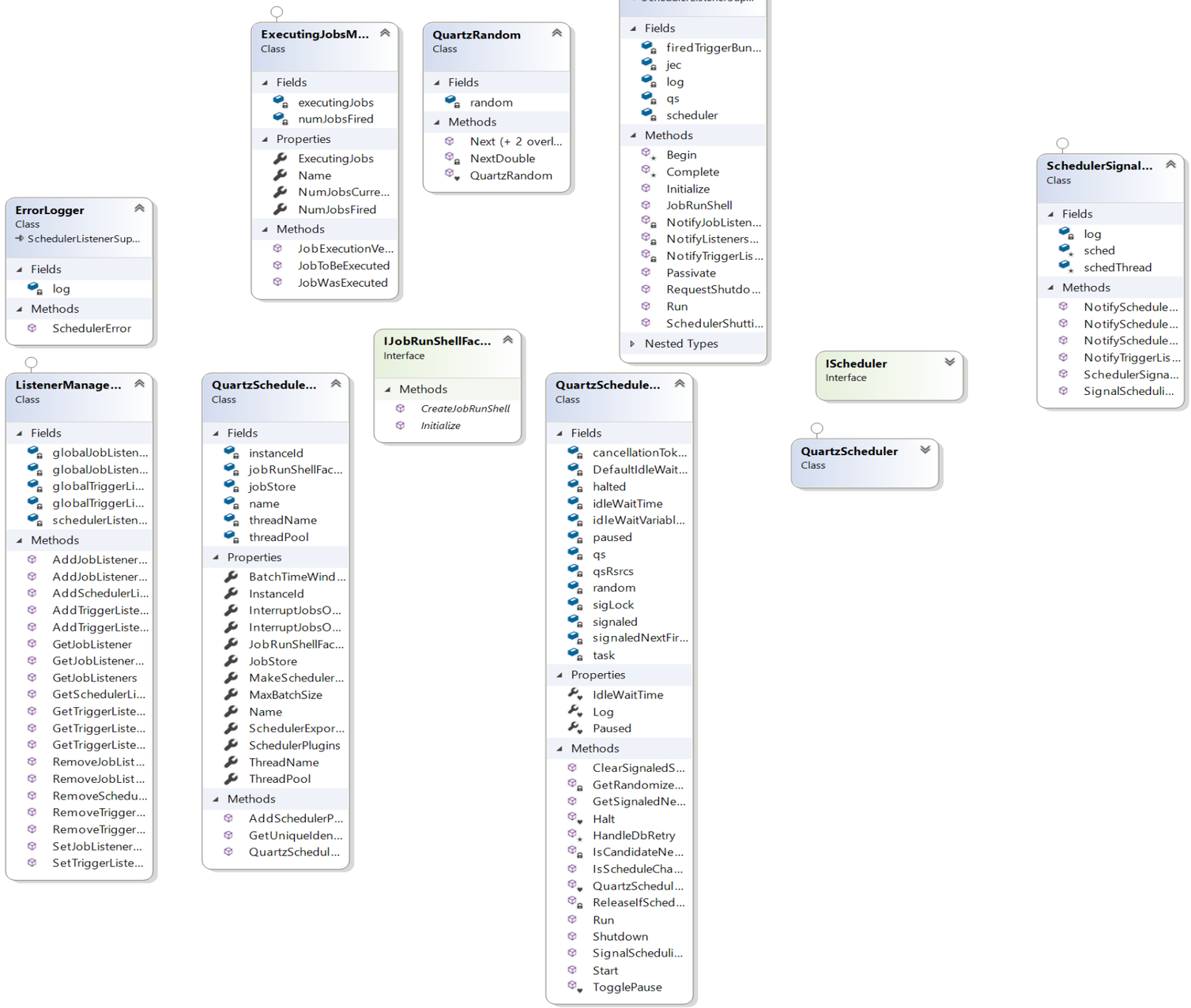
IT-74.191393.06.CCKM

Демонстраційний плакат «Схема структурна класів інтеграції з БД» до магістерської дисертації «Розробка планувальника задач до розподілених Enterprise систем»

Літ.	Маса	Мірило
Лист	Листів	

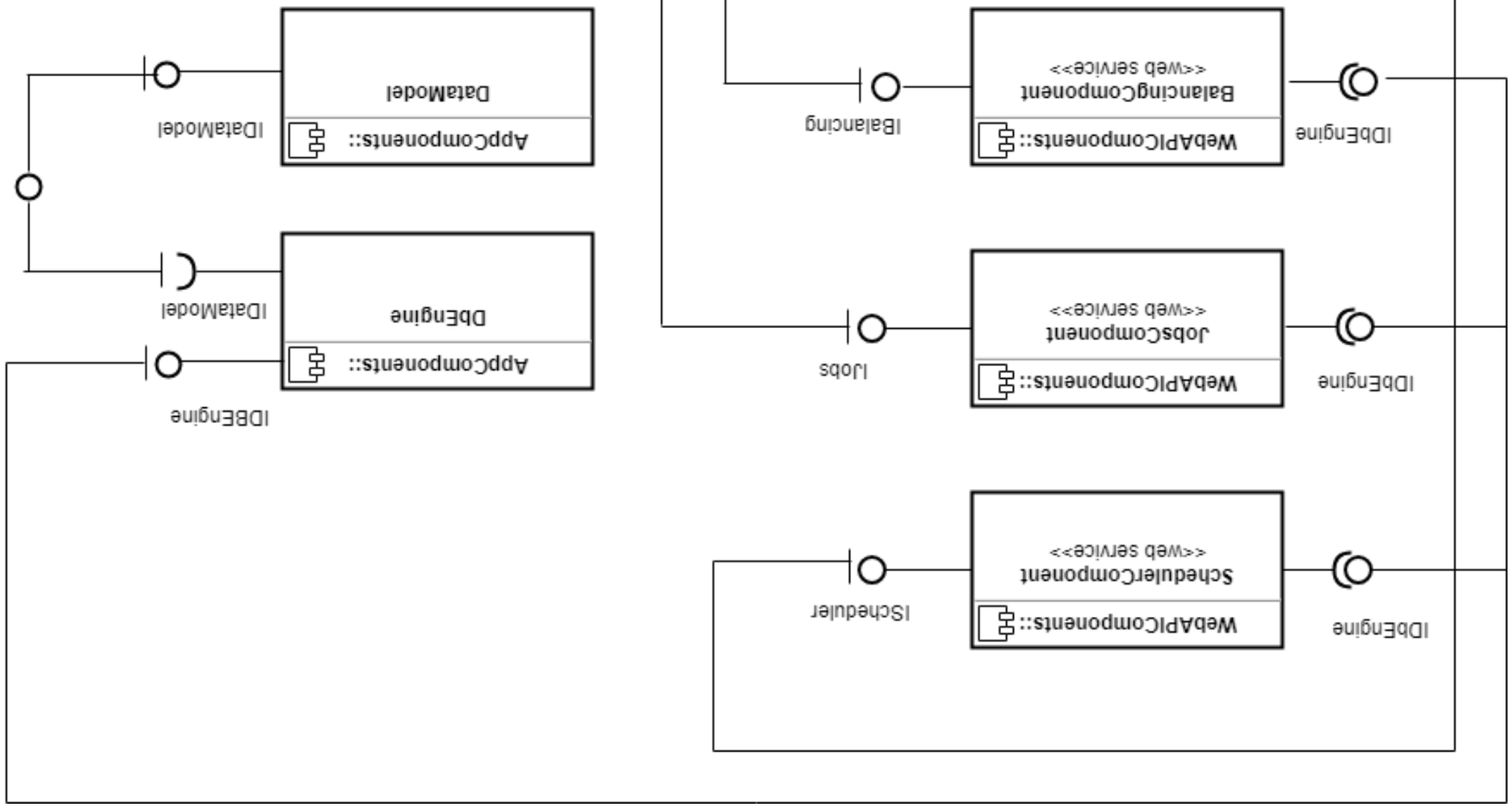
Кафедра Технічної кібернетики

Група IT-74



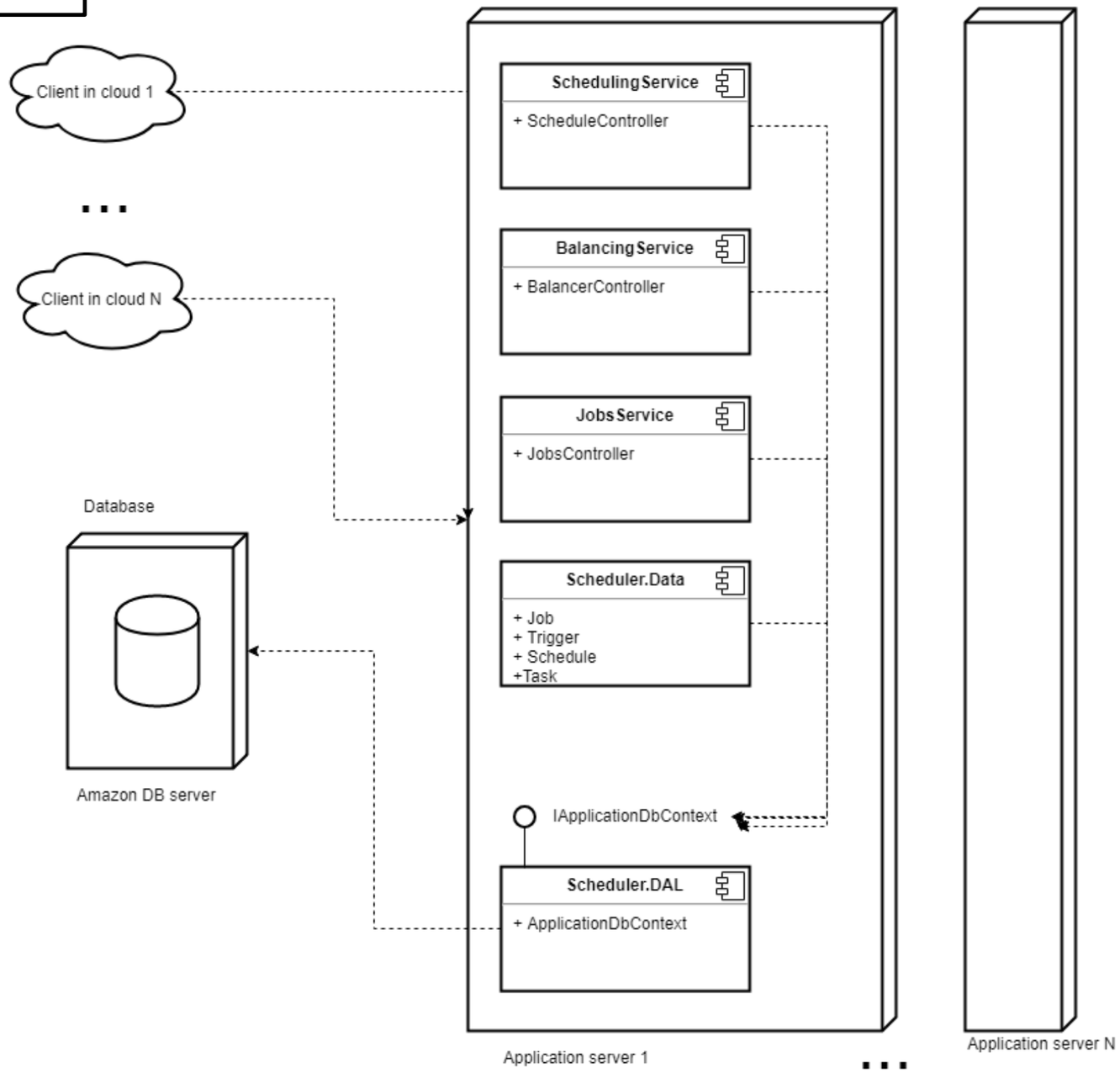
					IT-74.191393.07.ССКП	
Зм.	Лист	№ докум	Підпис	Дата	*Демонстраційний плакат «Схема структурна класів Core частини» до магістерської дисертації «Розробка планувальника задач до розподілених Enterprise систем»*	Літ. / Маса / Мірило
Розроб.		Хижняк М.Д.				
Перев.		Пасько В.П.				Лист / Листів
Н. кон.		Пасько В.П.			*Кафедра Технічної кібернетики*	*Група IT-74*
Затв.						

Підпис і дата | Інв. № дубл. | Взам. інв. № | Підпис і дата | Інв. № ориг.



Інв. № ориг.	Підпис і дата	Взам. інв. №	Інв. № дубл.	Підпис і дата

Зм.	Лист	№ докум	Підпис	Дата	ІТ-74.191393.08.ССК		
Розроб.	Хижняк М.Д.				Демонстраційний плакат «Схема даталогічної моделі даних» до магістерської дисертації «Розробка плагувальника задач до розподілених Enterprise систем»	Літ.	Маса	Масшт.
Перев.	Пасько В.П.							
Н. конт.	Пасько В.П.					Лист	Листів	
Затв.					Кафедра Технічної кібернетики	Група ІТ-74		



					ІТ-74.191393.09.ССР	
Зм.	Лист	№ докум	Підпис	Дата	*Демонстраційний плакат «Схема структурна розгортування» до магістерської дисертації «Розробка планувальника задач до розподілених Enterprise систем»*	Літ. / Маса / Мірило
Розроб.		Хижняк М.Д.				
Перев.		Пасько В.П.				Лист / Листів
Н. кон.		Пасько В.П.			*Кафедра Технічної кібернетики*	*Група ІТ-74*
Затв.						

Підпис і дата | Інв. № дубл. | Взам. інв. № | Підпис і дата | Інв. № ориг.

ВІДГУК
наукового керівника магістерської дисертації
на здобуття ступеня магістра

виконаної на тему: Розробка планувальника завдань для розподілених Enterprise систем

студентом (-кою) Хижняком Микитою Дмитровичем
(прізвище, ім'я, по батькові)

Актуальність теми обумовлюється в розробці точного алгоритму не лише розподілу завдань між вузлами системи але і алгоритму синхронізації результатів обчислень або обробки даних з різних вузлів системи.

Під час роботи над дисертацією проведено аналіз існуючих алгоритмів планування та варіантів їх використання в системах різного рівня навантаження, порівняння їх сильних та слабких сторін, створення на базі досліджень програмного продукту, що є абсолютно новим рішенням з точки зору оптимізації процесу планування завдань у розподілених системах Enterprise сегменту.

Тема дисертації «Розробка планувальника завдань для розподілених Enterprise систем» розкрита в повній мірі. Розроблено програмну реалізацію, яка демонструє нові можливості у предметній області. Під час роботи автор достатньо ознайомився зі структурою та принципом роботи планувальників задач, можливими складностями роботи планувальників у розподілених системах. Проявив достатньо високий рівень підготовленості при прийнятті сучасних технологічних рішень, уміння аналізувати літературні джерела, наполегливість у дослідженнях.

У цілому вважаю, що за своїм рівнем магістерська дисертація відповідає вимогам до атестаційних робіт ступеня магістра, а її автор, Хижняк М.Д. заслуговує присвоєння освітнього ступеня магістра за спеціальністю 121 «Інженерія програмного забезпечення» та кваліфікації магістр з інженерії програмного забезпечення.

Керівник магістерської дисертації

доц. к.т.н.		Пасько В.П.
(посада, науковий ступінь, вчене звання)	(підпис)	(ініціали, прізвище)

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
(назва інституту/факультету)

ЗАТВЕРДЖУЮ
Завідувач кафедри
технічної кібернетики_____
(назва кафедри)
_____ І.Р. Пархомей
(підпис) (ініціали, прізвище)
« 31 » жовтня 2018 р.

ІНДИВІДУАЛЬНИЙ НАВЧАЛЬНИЙ ПЛАН ПІДГОТОВКИ МАГІСТРА

за освітньо-професійною програмою «Програмне забезпечення інтелектуальних та робототехнічних систем» зі спеціальності 121 «Інженерія програмного забезпечення»

студента групи ІТ-74мп

Хижняк Микита Дмитрович
(прізвище, ім'я, по батькові)

1. Зарахований наказом ректора від 20 серпня 2018 р. № 2502-с
2. Термін навчання з 01 вересня 2018 р. до 30 грудня 2019 р.
3. Науковий керівник доц. к.т.н. Пасько В.П.
(науковий ступінь, вчене звання, прізвище та ініціали)

4. Тема стартап-проекту «Розробка планувальника завдань для розподілених Enterprise систем»

5. Науковий керівник і тема стартап-проекту ухвалені рішенням кафедри технічної кібернетики протокол № 3 від 31 жовтня 2018 р.

План роботи на перший рік навчання

№ п/п	Назви навчальних дисциплін	Кількість кредитів ЄКТС	Форма звітності	Відмітка наукового керівника
I семестр				
1	Практикум з іншомовного професійного спілкування	1,5	реферат	
2	Маркетинг стартап проектів	3	залік	
3	Наукова робота за темою магістерської дисертації -1. Основи наукових досліджень	2	залік	
4	Сучасні технології розроблення програмного забезпечення - 1	6	екзамен	
5	Методи проектування нереляційних баз даних	5	залік	
6	Методи експертних оцінок в компбтеризованих системах прийняття рішень - 1.	4	екзамен	
7	Методи інтелектуальної обробки даних	4	залік	
8	Менеджмент проектів програмного забезпечення	3,5	екзамен	
II семестр				

1	Інтелектуальна власність та патенознавство - 1. Право інтелектуальної власності	1		
2	Інтелектуальна власність та патенознавство - 2. Патентознавство та набуття прав	2	залік	
3	Основи сталого розвитку	2	залік	
4	Практикум з іншомовного професійного спілкування	1,5	залік	
5	Наукова робота за темою магістерської дисертації – 2. Науково-дослідна робота за темою магістерської дисертації	2	залік	
6	Сучасні технології розроблення програмного забезпечення - 2	5	залік	
7	Структурно-функціональний аналіз складних ієрархічних систем	7	екзамен	
8	Управління проектами та програмами	3	екзамен	
9	Технології електронної комерції	2,5	залік	
10	Проектування захищеного програмного забезпечення	3	екзамен	

Зміни (доповнення) до плану:

*Тема магістерської дисертації
,
затверджена наказом ректора від «____» _____ 20 ___ р. № _____

Науковий керівник _____ Магістрант_____

Звіт магістранта за перший рік навчання:

Рішення засідання кафедри, протокол № ___ від _____ 20 __ р.

Секретар кафедри _____ Завідувач кафедри _____

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
КАФЕДРА ТЕХНІЧНОЇ КІБЕРНЕТИКИ

ПЛАНУВАЛЬНИК ЗАВДАНЬ ДЛЯ РОЗПОДІЛЕНИХ ENTERPRISE-СИСТЕМ

ВИКОНАВ: СТУДЕНТ ГРУПИ ІТ-74МП
ХИЖНЯК МИКИТА ДМИТРОВИЧ
НАУКОВИЙ КЕРІВНИК: КАНДИДАТ ТЕХНІЧНИХ НАУК, ДОЦЕНТ КАФЕДРИ ТЕХНІЧНОЇ КІБЕРНЕТИКИ ФАКУЛЬТЕТУ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ НТУУ «КПІ» ІМЕНІ ІГОРЯ СІКОРСЬКОГО ПАСЬКО В.П.

Цілі

Скорочення часових витрат на виконання завдань в РСОД та покращення обробки та планування завдань у системах з розподіленою архітектурою для різних платформ.

Задачі

Аналіз існуючих методів планування завдань обробки даних в РСОД.

Розробка математичної моделі планування завдань в РСОД.

Розробка методики планування завдань в РСОД.

Програмна реалізація методів планування завдань у РСОД у вигляді планувальника завдань для розподілених Enterprise-систем.

Методологія дослідження

Об'єктом дослідження є системи планування завдань в РСОД, засоби та методи планування у розподілених системах.

Аналогічні системи, які стали основою для розробки даної системи, а саме Hangfire, Quartz.NET.

Теорія алгоритмів та математична статистика в якості методів дослідження.

Підходи та шаблони побудування мікросервісних розподілених застосувань на платформі .net core.

Алгоритми балансування розподілених систем.

NCron 

+ Зручне та гнучке API для розгортування завдань.

+ Можливість встроювати розклади завдань у збірку проекту, або ж зберігати їх у базі даних чи конфігураційних файлах.

+ Гнучке та зручне API журналювання задач.

+ Підтримка IoC контейнерів при розробці.

-Підтримка лише Windows платформи.

-Неможливість розгорнути планувальник у якості контейнера.

-Не підтримує роботу у розподілених системах.

- Відсутність широкого набору засобів контролю за алгоритмом планування.

Hangfire

+ Зручне та гнучке API для розгортування завдань.

+ Підтримка декількох ОС.

+ Підтримка контейнеризації при розгортуванні та розробці.

- Не підтримує роботу у розподілених системах.

- Складна архітектура планувальника, що потребує додаткових вкладень.

- Відсутність широкого набору засобів контролю за алгоритмом планування.

Quartz 

+ Набір властивостей та функцій для “Enterprise” систем.

+ Є порт Java планувальника під .NET платформу.

+ Підтримка контейнеризації при розгортуванні та розробці.

+ Підтримує роботу в розподілених системах, але має ряд обмежень щодо архітектури таких систем.

- Обмеження в архітектурі розподілених систем, що підтримуються.
- Відсутність широкого набору засобів контролю за алгоритмом планування.

- Більш складний API налаштування та журналювання завдань.

Опис алгоритму планування

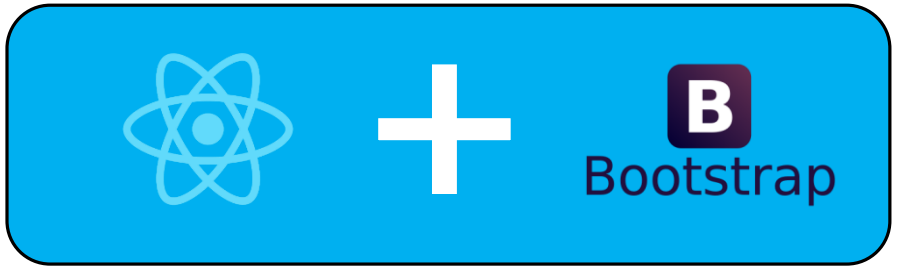
Алгоритм передбачає послідовність життєвих циклів популяції - поколінь. Кожен цикл складається з наступних операцій:

- селекція найбільш пристосованих особин;
- вибір батьківських пар;
- розмноження батьківських пар;
- мутація батьківських пар;

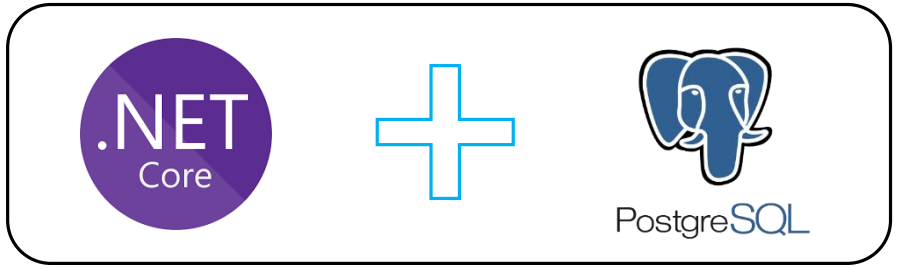
Процес повторюється до тих пір, поки на протязі заданої кількості поколінь не буде з'являтися особа с кращим значенням функції пристосованості.

Засоби розробки

Основні технології розробки frontend частини

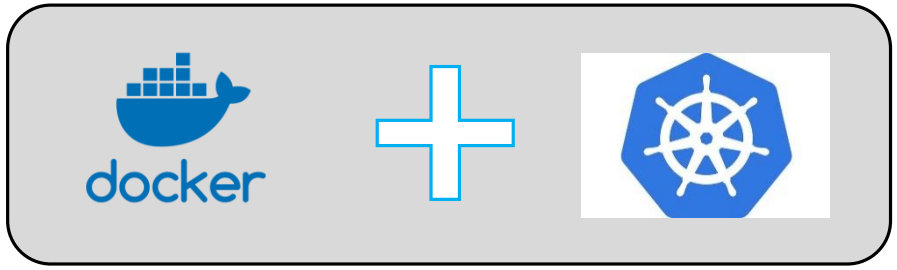


Основна платформа розробки



База даних

Архітектура розгортування



Архітектура планувальника

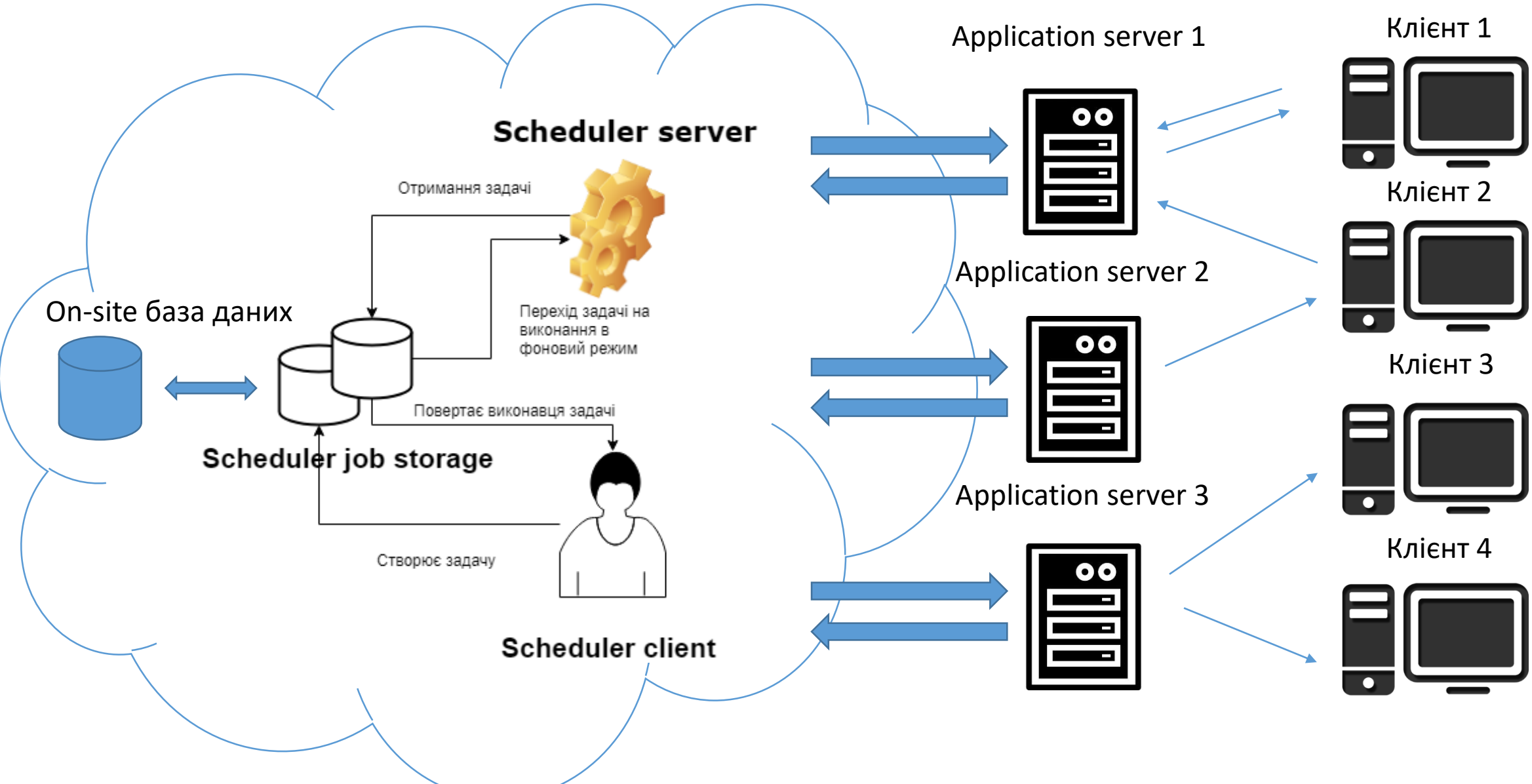
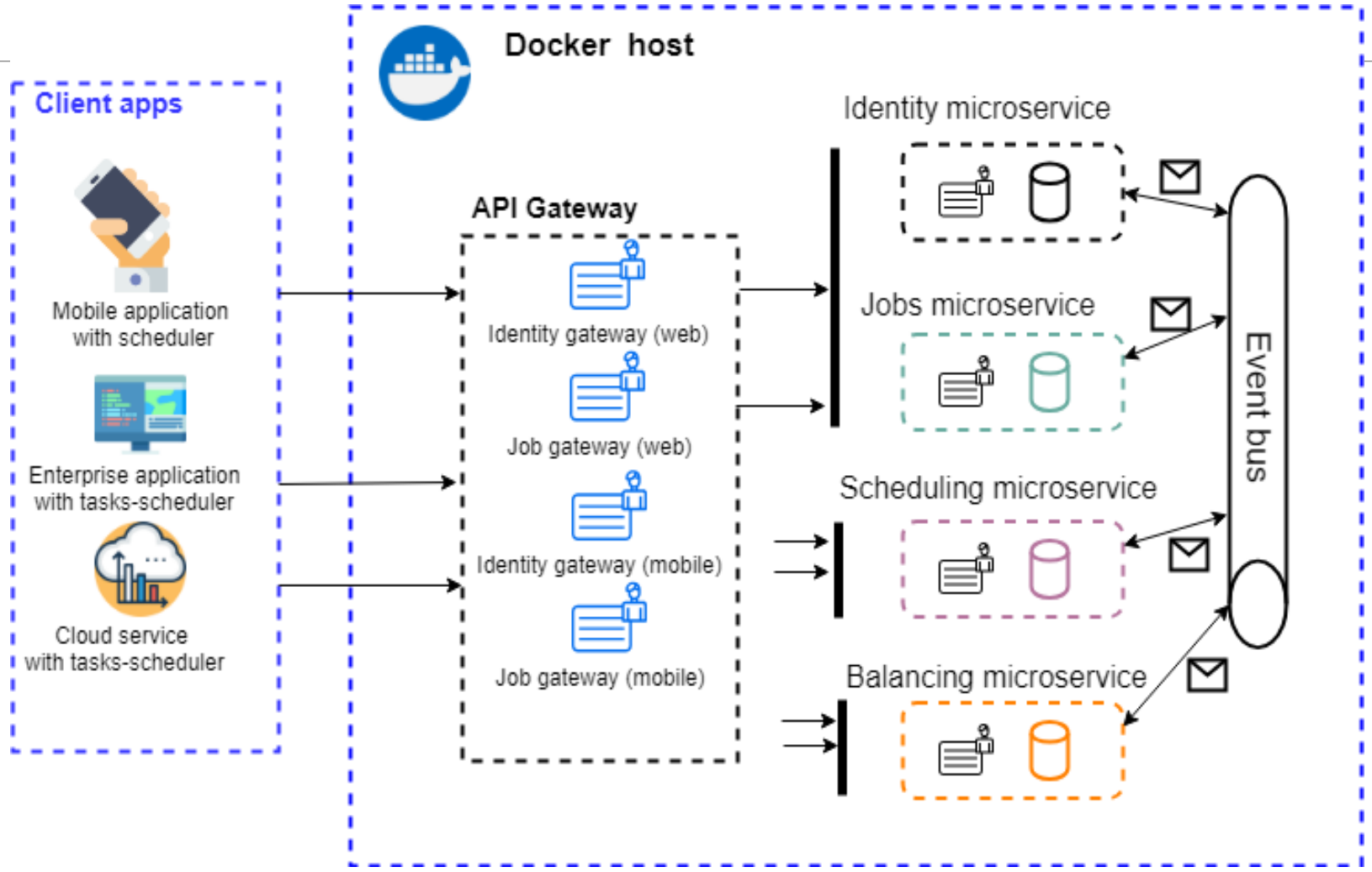
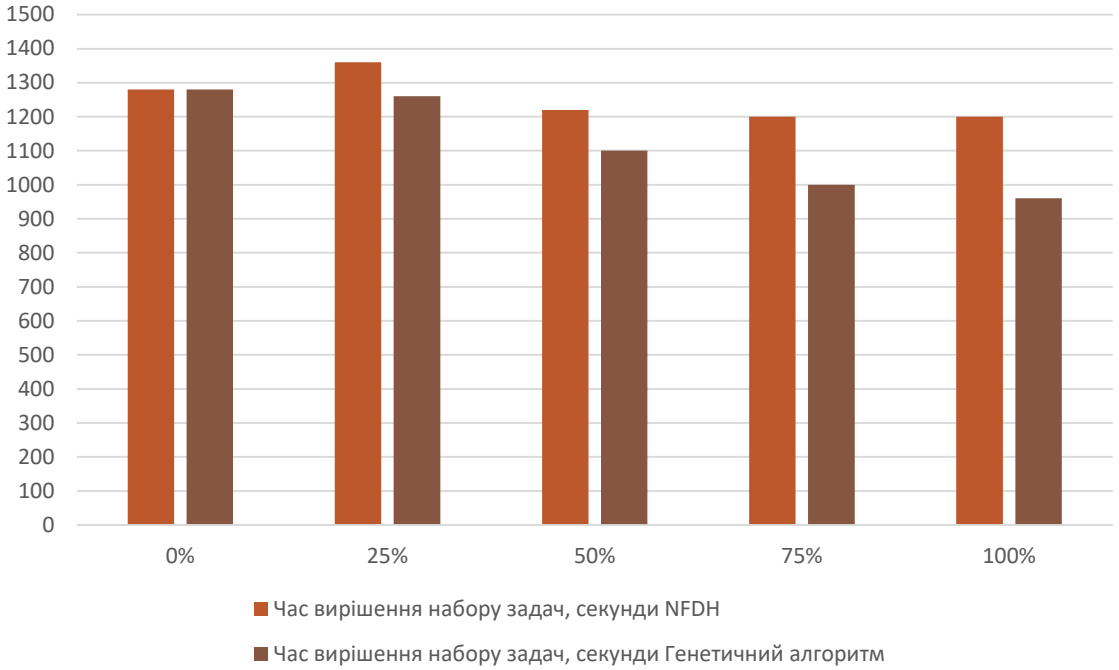
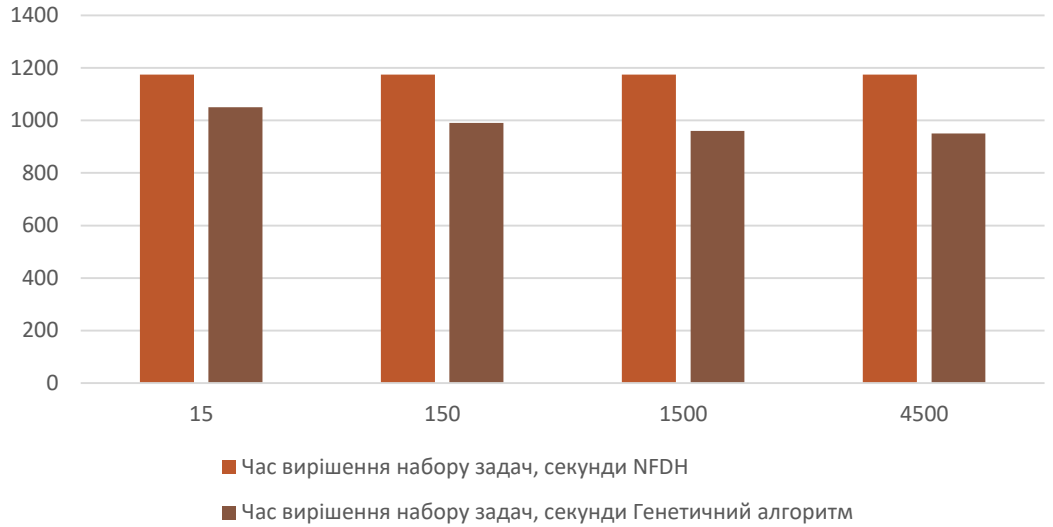


Схема мікросервісів

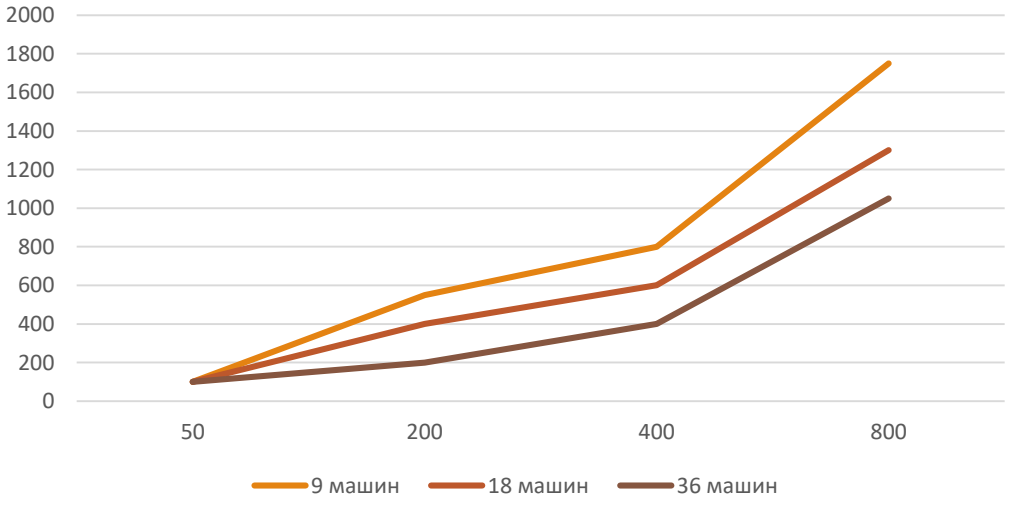






Залежність часу рішення набору задач від розміру популяції

Порівняння результатів роботи генетичного алгоритму та алгоритму NFDH



Залежність часу роботи генетичного алгоритму від масштабу системи

Microsoft Azure

Поиск ресурсов, служб и документов

Создать ресурс
Панель мониторинга
Все службы
ИЗБРАННОЕ
Все ресурсы
Группы ресурсов
Службы приложений
Приложения-функции
Базы данных SQL
Azure Cosmos DB
Виртуальные машины
Балансировщики нагрузки
Учетные записи хранения
Виртуальные сети
Azure Active Directory
Монитор
Advisor
Центр безопасности
Управление затратами + ...
Справка и поддержка

Панель мониторинга

Создать панель мониторинга | Отправить | Скачать | Изменение | Под

Все ресурсы

Все подписки

Обновить

tonightwebapi	Служба приложен...
tonightadministratorpanel	Служба приложен...
TonightWebAPI20181114032...	План служб прило...

Начало работы с Azure стало еще проще!

Запуск выбранного приложения в Azure всего за несколько шагов

Создать проект

Краткие руководства и учебники

Виртуальные машины Windows
Подготовка виртуальных машин Windows Server, SQL Server, SharePoint

Виртуальные машины Linux
Подготовка виртуальных машин Ubuntu, Red Hat, CentOS, SUSE, CoreOS

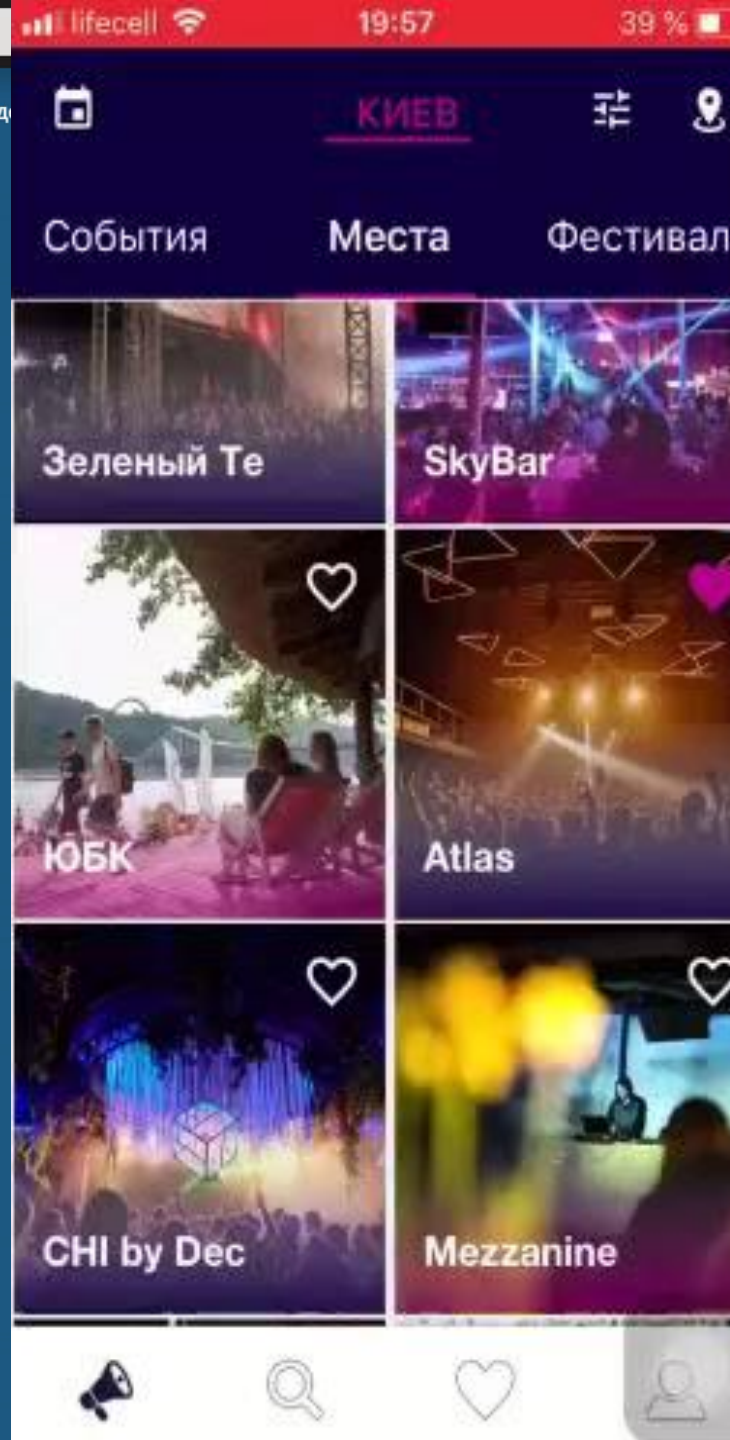
Служба приложений
Создавайте веб-приложения с помощью .NET, Java, Node.js, Python, PHP

Функции
Обработка событий с помощью архитектуры кода без сервера

База данных SQL
Управляемая реляционная база данных SQL как услуга

Работоспособность служб

Marketplace



Висновки

- Проведено огляд та аналіз предметної області планувальників завдань в системах, розглянуто їх основні типи та підходи до реалізації, а також проаналізовано можливості використання планувальників у Enterprise системах з розподіленою архітектурою;
- Розроблено генетичний алгоритм побудови оптимального розкладу виконання завдань;
- Створено бібліотеку планувальника на платформі .NET Core, що дозволяє досягнути умов cross-платформеності;
- Виконано порівняння існуючих алгоритмів та розробленого генетичного алгоритму за різних умов виконання задачі (кількість задач, відсоток масштабованих задач, кількість серверів у web-фермі);